Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit (Informatik)

Erstellen von Capture The Flag Challenges im Bereich Websicherheit zum Thema Web Cache Poisoning

Autoren	Dario Haas Timo Nigg
Hauptbetreuung	Dr. Ariane Trammell
Nebenbetreuung	Thomas Sutter

Datum	09.06.2023

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Name Studierende:**
Winterthur, 09.06.2023	Timo Nigg
Winterthur, 09.06.2023	Dario Haas

Zusammenfassung

Die ZHAW besitzt eine eigene CTFd Plattform, auf welcher CTF Challenges zur Verfügung gestellt werden. In dieser Arbeit wurden drei CTF Challenges zum Thema Web Cache Poisoning entwickelt, welche nun auf dieser Plattform verfügbar sind. Für die Informationsbeschaffung und Planung der Challenges wurde eine Marktanalyse durchgeführt. Diese untersuchte, wie andere Hochschulen CTF Challenges verwenden, wie diese aufgebaut werden und ob sie auch bereits welche zum Thema Web Cache Poisoning anbieten. Die aus der Umsetzung resultierenden Challenges besitzen alle eine eigene Schwierigkeitsstufe. Es gibt ein Tutorial als Einführung in das Thema. Dort werden die Grundlagen des Web Cache Poisoning an die Spielenden vermittelt. Bei der Medium Challenge können die Spielenden ihre Fertigkeiten schärfen, in dem sie das Gelernte aus dem Tutorial ohne weitere Hilfestellung anwenden. Die letzte und schwierigste Challenge bezieht sich auf eine in der Praxis gefundene Sicherheitslücke. Dabei können die Spielenden ihr Können unter Beweis stellen. Die ersten beiden Challenges konnten auch bereits im kleinen Rahmen an einem Event getestet werden. Das daraus resultierende Feedback floss in die Entwicklung mit ein und half bei der Verbesserung der Challenges.

Abstract

The ZHAW has its own CTFd platform where CTF challenges are made available. In this project, three CTF challenges on the topic of Web Cache Poisoning were developed and are now accessible on this platform. For gathering information and planning the challenges, a market analysis was conducted. This analysis examined how other universities use CTF challenges, how they are structured, and whether they already offer challenges related to Web Cache Poisoning. The challenges resulting from the implementation have different difficulty levels. There is a tutorial that serves as an introduction to the topic, providing players with the basics of Web Cache Poisoning. In the medium challenge, players can sharpen their skills by applying what they learned in the tutorial without further assistance. The final and most difficult challenge is based on a real-world security vulnerability. Players can demonstrate their abilities in this challenge. The first two challenges were also tested in a small-scale event. The feedback received from the testing was incorporated into the development process and helped improve the challenges.

Vorwort

Web Cache Poisoning ist eine bedeutende, aber eher unbekannte Sicherheitslücke. Die Entscheidung, uns mit Web Cache Poisoning zu befassen, basierte auf mehreren Überlegungen. Zum einen wollten wir uns mit einer weniger bekannten Sicherheitslücke befassen, die dennoch eine erhebliche Bedrohung darstellt. Die Komplexität und die verschiedenen Möglichkeiten von Web Cache Poisoning haben unser Interesse geweckt und uns motiviert, uns tiefergehend damit zu beschäftigen.

Ohne Unterstützung wäre diese Arbeit nicht möglich gewesen. Wir bedanken uns daher ganz herzlich bei folgenden Personen:

- Ariane, die uns ermöglicht hat, diese spannende Arbeit durchzuführen und ihre stetige Unterstützung bei Fragen und Problemstellungen.
- Thomas, der uns technisch zur Seite stand und des Öfteren den Challenge-Server neustarten durfte, da wir ihn an seine Grenzen brachten.
- Allen Korrekturlesenden für die Zeit, welche sie beim Korrekturlesen aufgewendet haben.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Ausgangslage . . . 1
1.2 Stand der Technik . . . 1
1.3 Motivation . . . 1
1.4 Aufgabenstellung und Zielsetzung . . . 2
1.5 Projektplan . . . 2
1.6 Überblick . . . 3

2 CTF Challenges in Higher Education **4**
2.1 Verwendungszweck . . . 4
2.2 Technische Aspekte von CTF Challenges . . . 5

3 Theoretische Grundlagen **7**
3.1 Cache . . . 7
3.2 Cache-Key . . . 7
3.3 Web Cache Poisoning . . . 8
3.4 Angriffsmuster . . . 9
3.4.1 Cache Oracle . . . 9
3.4.2 Key Handling . . . 10
3.4.3 Exploit . . . 10
3.4.4 Cachebuster . . . 10
3.5 Gegenmassnahmen . . . 11

4 Anforderungen **13**
4.1 Nichtfunktionale Anforderungen . . . 13
4.2 Technische Anforderungen . . . 14
4.3 ZHAW Umgebung . . . 15
4.4 Sicherheit . . . 15

5 Konzept **17**
5.1 Challenges . . . 17
5.1.1 Easy: How to Web Cache Poisoning . . . 17
5.1.2 Medium: Cross Site Cookie Mining . . . 18
5.1.3 Hard: Double Cache Mayhem . . . 20
5.2 Webseite . . . 21
5.2.1 Aufbau der Webseite . . . 21
5.2.2 Entscheidungsfaktoren . . . 23

6 Architektur **24**
6.1 Verwendete Technologien . . . 24
6.2 Integration in die ZHAW CTFd Infrastruktur . . . 25
6.3 Sicherung der Challenges vor Fremdeinwirkung . . . 27

6.4 Challenges . . . 29
6.4.1 Allgemeiner Aufbau . . . 29
6.4.2 Easy: How To Web Cache Poison . . . 34
6.4.3 Medium: Cross Site Cookie Mining . . . 36
6.4.4 Hard: Double Cache Mayhem . . . 42
6.5 Testkonzept . . . 46
6.5.1 Komponententest . . . 46
6.5.2 Integrationstest . . . 47
6.5.3 Systemtest . . . 48

7 Verifizierung und Evaluation **49**
7.1 Verifizierung der Anforderungen . . . 49
7.2 Evaluation der Challenges Easy und Medium . . . 51

8 Ausblick **52**

9 Konklusion **53**

Literaturverzeichnis **54**

Abbildungsverzeichnis **56**

Tabellenverzeichnis **57**

Codeverzeichnis **58**

A Anhang **59**
A.1 API-Documentation . . . 60
A.2 Konfiguration für die Challenges auf der CTFd Plattform der ZHAW . . . 67
A.3 Resultate der Umfrage . . . 68

1 Einleitung

Die ZHAW besitzt eine eigene Plattform, auf welcher sie Capture The Flag (CTF) Challenges anbietet. Jedoch gibt es noch keine richtigen Challenges, die gespielt werden können. Mit dieser Arbeit soll dies geändert werden.

1.1 Ausgangslage

CTF Challenges sind Softwareaufgaben, bei welchen sogenannte Flags innerhalb einer Software versteckt werden und diese dann von den spielenden Personen wieder gefunden werden müssen. CTF Challenges werden für verschiedene Zwecke eingesetzt. Einerseits werden sie innerhalb des Bildungsbereichs verwendet, um verschiede Aspekte in der Softwaresicherheit zu trainieren und andererseits können sie auch einfach als Spiel angesehen werden, welches Spass bereiten soll. Neben der Bildung und dem Spass werden CTF's auch regelmässig in Turnieren gespielt, wo verschiedene Teams gegeneinander antreten und versuchen, verschiedenen Challenges zu lösen. Diese Challenges werden hauptsächlich von Sicherheitsexpert*innen oder von Sicherheitsinteressierten, welche eine gewisse Erfahrung und Affinität im Bereich der Softwaresicherheit aufweisen, erstellt beziehungsweise gespielt. Ebenfalls werden diese von Informatik Studierenden an Hochschulen verwendet, um das Gelernte praktisch anzuwenden und zu trainieren.

1.2 Stand der Technik

Wie bereits erwähnt besitzt die ZHAW eine CTF Plattform. Diese Plattform basiert auf dem CTFd Framework [1]. Die CTF Plattform besitzt drei Kategorien von Challenges: Websicherheit, Kryptografie und Forensik. Für diese sind momentan lediglich Teaser-Challenges verfügbar, die gespielt werden können. Challenges die sich mit konkreten Sicherheitslücken in der jewiligen Kategorie auseinandersetzen, sind noch aussstehend.

Die CTF-Plattform der ZHAW ist über die URL `ctf.cloudlab.zhaw.ch` erreichbar. Es ist jedoch zu beachten, dass diese Seite nur im internen Netzwerk der ZHAW zugänglich ist.

1.3 Motivation

Es gibt viele sehr bekannte Arten von Web-Angriffen, wie zum Beispiel SQL-Injection und XSS. Web Cache Poisoning gehört (noch) nicht dazu. Um eine grosse Menge von Anfragen zu bewältigen, sind Caches hervorragend für Load Balancing geeignet. Bei falscher Anwendung sind sie jedoch ein erstklassiger Träger für bösartige Responses. Aus diesem Grund haben wir uns für Web Cache Poisoning als Grundlage für die CTF Challenges entschieden.

1.4 Aufgabenstellung und Zielsetzung

Das Ziel dieser Arbeit war es, Web Cache Poisoning CTF-Challenges zu erstellen, welche auf der CTF Plattform der ZHAW gehostet werden können. Damit diese Challenges erfolgreich umgesetzt werden konnten, wurde zuvor eine Marktanalyse durchgeführt. Bei dieser sollte herausgefunden werden:

- Wie werden CTF Challenges im Bereich Higher Education verwendet?
- Gibt es bereits Web Cache Poisoning CTF Challenges und wenn ja, wie werden sie eingesetzt?

Das Beantworten dieser Fragen sollte dazu beitragen, dass die eigenen CTF Challenges erfolgreich umgesetzt werden konnten und ausserdem, dass sie Spass machen beim Spielen.

Die genaue Aufgabenstellung lautete wie folgt:

- Erstellen von eigenen Web Cache Poisoning CTFs für die CTF Plattform der ZHAW.
- Durchführung einer Marktanalyse:
  1. Analyse der Verwendung von CTF Challenges im Bereich Higher Education.
  2. Untersuchen, was für Web Cache Poisoning CTF Challenges es bereits gibt und wie diese eingesetzt werden.

1.5 Projektplan

Für die Planung der abzuarbeitenden Aufgaben wurde ein Projektplan erstellt. In diesem Abschnitt wird der Aufbau dieses Plans kurz beschrieben.

Wie in Abbildung 1.1 zu sehen ist, wurde grundsätzlich in iterativen Sprints mit einer Dauer von zwei Wochen gearbeitet. Die Wahl der Sprintlänge basiert auf den gesammelten Erfahrungen aus vorgängigen Semesterprojekten und hat sich bisher bewährt.

Wie bei einem iterativen Vorgehen üblich, wurden Reviewsitzungen abgehalten. Diese erfolgten wöchentlich. Bei diesen Sitzungen wurden die erledigten Arbeiten sowie das weitere Vorgehen mit den betreuenden Dozierenden besprochen.

		Kalenderwoche														
Anfang	**Ende**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**Tätigkeit**
28.02.23	06.03.23																Themenbestimmung
07.03.23	20.03.23																Literaturrecherche
21.03.23	03.04.23																Requirements, Architektur
04.04.23	29.05.23																Implementieren
02.05.23																	CTF Event
30.05.23	08.06.23																Dokumentation
09.06.23																	Abgabe

ABBILDUNG 1.1: Zeitplan für die Bachelorarbeit.

Das CTF Event, wie in Abbildung 1.1 zu sehen ist, war der erste Meilenstein. Bis zu diesem Zeitpunkt mussten die ersten beiden Challenges spielbar sein. Da dort Feedback für die erstellten Challenges eingeholt werden konnte. Der zweite Meilenstein ist die Abgabe der Arbeit, bis dahin muss der Bericht so wie die Implementierung der Challenges abgeschlossen worden sein.

Zu bemerken gibt es, dass im Zeitplan lediglich zwei Wochen für das Schreiben des Berichts eingezeichnet sind. Es wurde auch vor diesen zwei Wochen am Bericht gearbeitet, jedoch sind diese explizit für den Bericht eingeteilt und es fand ein Code-Freeze statt.

1.6 Überblick

Dieser Abschnitt dient als kleine Übersicht über die ganze Arbeit.

CTF Challenges in Higher Education

Die Nutzung und der Aufbau von CTF Challenges im Bereich der Higher Education wird untersucht.

Theoretische Grundlagen

Vermittelt die Theorie eines Web Cache Poisoning Angriffs und erläutert die Schritte, welche bei einem Angriff unternommen werden.

Anforderungen

Die Anforderungen, welche an die Challenges gestellt werden, werden aufgelistet und beschrieben.

Konzept

Basierend auf den gestellten Anforderungen wird ein Konzept für die Challenges ausgearbeitet.

Architektur

Beschreibt den Aufbau und die Umsetzung der Challenges anhand der gestellten Anforderungen.

Verifizierung und Evaluation

Anhand der Anforderungen werden die Resultate verifiziert und evaluiert. Anforderungen, die nicht vollständig umgesetzt worden sind, werden genauer betrachtet und erörtert.

Ausblick

Erläutert, was für Arbeiten noch offen stehen und beschreibt, was in zukünftigen Arbeiten noch unternommen werden kann, um die Challenges zu verbessern oder zu erweitern.

Konklusion

Gibt einen kurzen Rückblick über die gesamte Arbeit.

2 CTF Challenges in Higher Education

Um die Nutzung von CTF Challenges im Bereich der Higher Education zu untersuchen und um mehr über den technischen Aufbau von CTF Challenges im Allgemeinen zu erfahren, wurde eine Marktanalyse durchgeführt. Die Beschaffung von Informationen zum Thema CTF Challenges im Bereich der Higher Education gestaltete sich relativ schwierig. Es gibt nicht viele wissenschaftliche Arbeiten, bei welchen die Verwendung von CTF Challenges in Higher Education aufgezeigt wird. Es gibt jedoch mehrere Arbeiten, welche analysieren, was für einen Einfluss spielbasierte Lernmethoden (z.B. CTF Challenges) auf die Studierenden haben. Im Gegensatz zu CTF Challenges im Bereich Higher Education gibt es für den Aufbau und die Architektur von CTF Challenges genügend bestehende Arbeiten, welche als Quellen verwendet werden können.

Die aus der Marktanalyse folgenden Erkenntnisse werden in den folgenden Abschnitten zusammengetragen.

2.1 Verwendungszweck

Es wurden zwei Hauptaspekte im Bereich der Higher Education untersucht. Einerseits, wie sich die Verwendung von CTF Challenges auf die Leistungen von Studierenden auswirkt und andererseits, welche technischen Aspekte die Erstellung von CTF Challenges für Hochschulen beeinflussen.

Da es verschiedene Formate von CTF Challenges gibt, als Grobkategorisierung «Jeopardy» und «Attack and Defence», müssen die Vor- und Nachteile der Formate unterschieden werden.

- **Jeopardy**

  Bei Jeopardy CTF Challenges werden einzelnen Teams, bestehend aus Studierenden, mehrere Aufgaben aus verschiedenen Bereichen wie Kryptografie, digitale Forensik, Systemsicherheit und Sicherheit von Webanwendungen gestellt [2]. Es steht ihnen frei, die Aufgaben in beliebiger Reihenfolge zu lösen, und sie arbeiten isoliert von anderen Teams.

- **Attack and Defence**

  Bei Attack and Defence CTF Challenges werden den Teams eine virtuelle Maschine zur Verfügung gestellt, die aus identischen Diensten besteht, welche alle die gleichen Schwachstellen aufweisen [2]. Das Ziel besteht darin, die bestehenden Sicherheitsprobleme zu entdecken, sie zu beheben und Dienste anderer Teams unter Ausnutzung der entdeckten Schwachstellen anzugreifen.

In der Bildung beziehungsweise im Allgemeinen werden mehr Jeopardy Style CTF Challenges als Attack and Defence CTF Challenges veranstaltet [2]. Dies obwohl davon ausgegangen wird, dass von den beiden Formaten Attack and Defence CTF Challenges aufgrund ihres interaktiven und realitätsnahen Charakters bessere Lernergebnisse liefern [2]. Dies kann dadurch begründet werden, dass die benötigte Infrastruktur und Rechenleistung für Attack and Defence CTF Challenges wesentlich höher ausfällt als bei Jeopardy CTF Challenges [2]. Zusätzlich können bei Jeopardy CTF Challenges einzelne Challenges mittels Berichten von Studierenden teilbewertet werden. Durch die Verwendung von Flags, die am Ende der Challenge übermittelt werden, wird eine automatisierte Bewertung ermöglicht. [3].

Als Beispiel für die Wirksamkeit von CTF Challenges als Lehrmethode kann die Untersuchung der TU Wien genommen werden. Dort setzt man seit 2012 auf CTF Challenges, um den Studierenden die Aspekte der IT-Security näher zu bringen [4]. Die TU Wien verwendet dabei das Jeopardy Format. Sie untersuchten, wie ihr pädagogischer Ansatz bei den Studierenden ankam und wie effektiv sich dieser als Lehrmethode erwies. Die Ergebnisse der Studie deuten darauf hin, dass die spielerische Kurserfahrung und der Wettbewerb unter den Studierenden ein effektiver Weg ist, um Sie zu motivieren, mehr Mühe und harte Arbeit in ihre Sicherheitsausbildung zu stecken. Darüber hinaus steigert der Ansatz auch ihr Interesse an der IT-Sicherheit und führt in der Folge zu einem hohen Wissenszuwachs [4].

Was dabei zu beachten ist, ist, dass die meisten Konzepte von CTF-Events nicht auf pädagogischen Theorien beruhen [3]. Darüber hinaus ist es aufgrund des Mangels an empirischen Daten in den meisten Studien nicht möglich, einen ausdrücklichen Zusammenhang zwischen den Merkmalen des Live-Wettbewerbs und bestimmten Bildungseffekten herzustellen [3].

2.2 Technische Aspekte von CTF Challenges

Auch bei den technischen Aspekten muss man zwischen einfachen Jeopardy und Attack and Defence CTF-Events unterscheiden, da diese auf jeweils ihre eigene Art aufgesezt, beziehungsweise verteilt werden müssen.

- **Jeopardy**

  Jeopardy Style CTF Challenges sind von der Architektur her flexibel und auf verschiedene Wege umsetzbar. Hier wird sich auf den Aufbau mittels CTFd beschränkt, da dies einerseits vom InIT der ZHAW zur Distribution von Challenges verwendet wird und zum anderen da aufgrund der grossen Anzahl an unterschiedlichen Architekturen nicht alle abgedeckt werden können. CTFd ist eine Open Source Plattform, worüber einzelne Challenges gehostet werden können. Um über CTFd Challenges zu hosten muss CTFd selbst auf einem Server aufgesetzt werden. Über CTFd aus können dann Challenges verlinkt werden. Diese Challenges sind selbst für die Skalierung und Distribution einzelner Images (Challenges) verantwortlich. Das heisst, dass CTFd eine «Sammlung» von verschieden Challenges ermöglicht, welche dann von den Spielenden aufgerufen werden können. Wie in Abbildung 2.1 zu sehen ist, verweist CTFd auf den Challengeserver, welcher für die Skalierung und Verteilung zuständig ist. Dieser verteilt dann auch die einzelnen Challenges an die spielenden Personen.

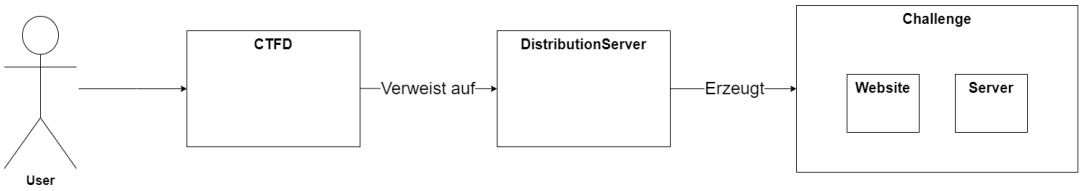


ABBILDUNG 2.1: Aufbau einer CTF-Challenge mittels CTFd.

- **Attack and Defence**

  Die Infrastruktur der meisten Attack and Defence CTF Challenges basiert auf dem von iCTF [5] über mehrere Ausgaben hinweg entwickelten Konzept, bei dem alle Dienste in eine virtuelle Maschine gepackt werden [2]. Dabei gibt es zwei unterschiedliche Ansätze: zum einen den dezentralisierten und zum anderen den zentralisierten Ansatz. Fast alle Online-Angriffs- und Verteidigungs-CTF-Veranstaltungen nutzen eine dezentrale Architektur. Die Infrastruktur besteht aus mehreren geografisch verteilten Teamnetzwerken und einem Organisatorennetzwerk, die über ein virtuelles privates Netzwerk (VPN) miteinander verbunden sind [2]. Im Gegensatz zur dezentralisierten Architektur steht die zentralisierte Architektur, wo das Organisationsteam des Events, die virtuellen Maschinen aller Teams auf ihren Servern hosten. Die Teams können dann mittels SSH auf die virtuellen Maschinen zugreifen. Dieser Ansatz reduziert den Aufwand für die technische Einrichtung, den die Teams betreiben müssen [2].

3 Theoretische Grundlagen

Im folgenden Kapitel wird auf die technischen Aspekte von Web Cache Poisoning eingegangen. Dabei werden die folgenden Aspekte erläutert:

- **Cache**

  Wie funktioniert ein Cache?

- **Web Cache Poisoning**

  Was bedeutet Web Cache Poisoning?

  Was sind die Ursachen und Folgewirkungen?

- **Angriffsmuster**

  Wie sieht ein Web Cache Poisoning Angriff aus?

- **Gegenmassnahmen**

  Was kann man unternehmen, um Angriffe zu verhindern?

3.1 Cache

Caches helfen, die Belastung in einem Netzwerk zu reduzieren. Dies wird erreicht indem ähnliche Requests von einem Cache erkannt werden, welcher die Response zwischenspeichert (engl. to cache).

Damit ein Cache überprüfen kann, ob er für eine einkommende Request bereits eine Response besitzt, muss er diese mittels eines Identifikators identifizieren können. Dies wird erreicht, indem für jede Request ein Cache-Key erzeugt wird.

Ein Cache kann es sowohl auf der Seite des Benutzers (Client-Side-Cache) geben als auch auf der Seite des Servers (Server-Side-Cache). Beim Client-Side-Chache werden Request/Response-Paare nur für einen*eine Benutzer*in gespeichert, während beim Server-Side-Caching Request/Response-Paare für mehrere Benutzer*innen gespeichert werden. Für Web Cache Poisoning ist nur der Server-Side-Cache von Interesse. Ist im Verlaufe des Berichts von einem Cache die Rede, dann bezieht sich dies auf den Server-Side-Cache.

3.2 Cache-Key

Ein Cache-Key ist ein Schlüssel zur eindeutigen Identifizierung von ähnlichen Requests. Für die Erzeugung eines solchen Schlüssels nimmt der Cache eine vordefinierte Anzahl Headers und speichert diese [6]. Wie viele Headers benutzt werden, d. h. welche für den Cache-Key benutzt werden, hängt von der Konfiguration des Caches ab. Beispielsweise können Host und URL einen Cache-Key bilden, wie das beim Codeausschnitt 3.1 der Fall ist.

```
GET /index.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 ... Firefox/57.0
Cookie: language=de;
Accept: .....
```

CODE 3.1: Einfaches Beispiel eines Cache-Keys.

Diese Mechanik kann jedoch bereits zu Cache Poisoning führen, wie der Codeausschnitt 3.2 zeigt. Denn ein Request, welcher von einem*einer Benutzer*in kommt, der*die eine andere Sprache spricht, erhält die falsche Response.

```
GET /index.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 ... Firefox/57.0
Cookie: language=en;;
Accept: .....
```

CODE 3.2: Einfaches Beispiel von Web Cache Poisoning.

Wie viele Headers in den Cache-Key aufgenommen werden sollen, ist sehr anwendungsspezifisch. Cloudflare bspw. verwendet standardmässig 13 Headers [7], um einen Cache-Key zu erstellen. Ein sorgfältiges Auslesen der Headers dient jedoch bereits zur Prävention von Web Cache Poisoning bei. Mehr dazu wird im Abschnitt 3.5 erläutert.

3.3 Web Cache Poisoning

Web Cache Poisoning ist ein Angriff auf den Cache einer Website. Ziel dabei ist es eine bösartige Response in den Cache zu legen, welche für eine bestimmte Request zurückgegeben wird. Dies geschieht indem der*die Angreifer*in nach Headers sucht, mit welchen der Server interagiert, die jedoch vom Cache nicht in den Cache-Key mit aufgenommen werden. Diese sogenannten unkeyed Headers öffnen dann ein Tor für potenzielle Angriffe.

Wird ein solcher unkeyed Header gefunden und der*die Angreifer*in schafft es, dass die Request im Cache gespeichert wird, sind eine Folge von möglicherweise verheerenden Angriffen zulässig. Dies hängt davon ab, wie genau der Server mit dem Header interagiert.

Ein realer Angriff, welcher aufgrund von Web Cache Poisoning möglich war, ist der «Drupal Open Redirect» [8] [9] [10]. Im Symfony-Framework wurde ein Problem entdeckt, das auf die Verwendung eines veralteten IIS-Headers zurückzuführen ist. Da das Symfony-Framework auch in Drupal verwendet wird, ist auch Drupal von dieser Sicherheitslücke betroffen.

Diese Lücke ermöglicht in Drupal, dass der Ort, zu dem ein Redirect führen soll, manipuliert werden kann. Dadurch eröffnet sich die Möglichkeit für Man-in-the-Middle (MITM)-Attacken. Ein Angreifer könnte beispielsweise Login-Seiten ersetzen und so Zugriff auf Benutzerdaten erlangen.

3.4 Angriffsmuster

Abbildung 3.1 visualisiert das Vorgehen bei einem Web Cache Poisoning Angriff.

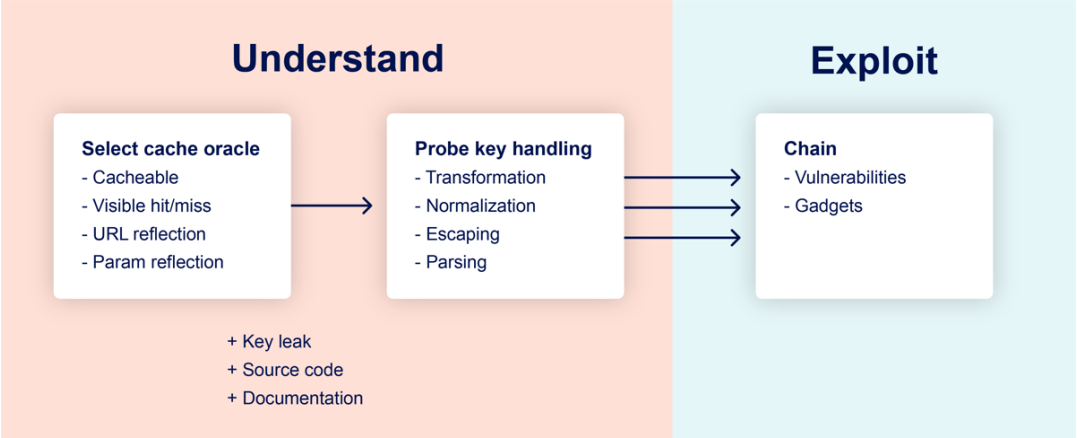


ABBILDUNG 3.1: Methodik eines Web Cache Poisoning Angriffs [11].

Dabei wird in drei Schritten vorgegangen.

1. **Cache Oracle**: Herausfinden, wie der zugrundeliegende Cache funktioniert.
2. **Key Handling**: Herausfinden, wie der Cache mit den verschiedenen Keys umgeht.
3. **Exploit**: Wissen bündeln und einen Angriff kreieren.

Beim ganzen Prozess ist es wichtig, dass ein Cachebuster benutzt wird (siehe Abschnitt 3.4.4).

Weitere wichtige Dokumente/Quellen, die bei einem Web Cache Poisoning Angriff berücksichtig werden können, sind:

1. **Durchsickern von Informationen:** Wenn Informationen über eine Applikation ungewollt veröffentlicht werden, kann dies ein potenzieller Einstieg für Web Cache Poisoning sein.
2. **Quellcode:** Es können Sicherheitslücken entdeckt werden, die noch undokumentiert sind oder noch nicht behoben wurden.
3. **Dokumentation:** Kann relevante Informationen über die Applikation besitzen. Auch ältere Dokumente sollten beachtet werden. Bspw. kann es passieren, dass eine frühere Version einer RestAPI ein Feature besitzt, dass es in einer neueren Version nicht mehr gibt. Wenn unsauber gearbeitet wurde, besteht die Möglichkeit, dass dieses Feature in der neuen Version immer noch zugänglich ist.

3.4.1 Cache Oracle

Wie bereits bei den Cache-Keys erwähnt, sind Caches individuell konfigurierbar. Um einen Web Cache Poisoning Angriff zu starten, muss man verstehen, wie der zugrunde liegende Cache funktioniert. Dazu wählt man ein Cache Oracle. Ein Cache Oracle ist ein Endpunkt (eine Webseite, Link, Download-Button, usw.). Mithilfe des Cache Oracles wird versucht herauszufinden, wie der Cache funktioniert [11].

Eine wichtige Voraussetzung ist dabei zu erkennen, ob eine Response vom Server oder vom Cache kommt. Cloudflare bspw. besitzt den Header CT-Cache-Status, welcher unbezahlbare Informationen besitzt. Der Header signalisiert, ob die Response vom Server (Status: MISS) oder vom Cache (Status: HIT) kam [12]. Ist dies nicht direkt in der Response erkennbar, so kann man alternativ auch die Antwortzeiten betrachten.

Idealerweise ist in der Response die URL oder ein bis mehrere Parameter enthalten [11]. Dazu mehr im Schritt Key Handling.

3.4.2 Key Handling

Ist das Cache Oracle fündig geworden, d. h. es gibt einen Endpunkt, der im Cache gespeichert wird, muss man als Nächstes herausfinden, ob die Response in irgendeiner Art und Weise manipuliert werden kann. Ziel dabei ist es zu schauen, wie sich zwei fast identische Requests verhalten [11].

Das Vorgehen ist dabei wie folgt:

1. Eine Request wird erzeugt und abgesendet.
2. Eine fast identische Request wird erzeugt und abgesendet. Ist die Antwort gleich wie bei der ersten Request, so wurde diese vom Cache erzeugt.

Beim Key Handling können bekannte Verarbeitungsverfahren angewendet werden, um fast identische Requests zu erzeugen. Mögliche Änderungen können durch Transformation, Normalisierung, Escaping oder Parsing erzeugt werden.

Am Schluss besitzt man folgende Informationen:

1. Eine Übersicht über alle Headers, die im Cache-Key vorhanden sind.
2. Eine Übersicht über alle Headers, die nicht vom Cache in den Cache-Key aufgenommen werden, die jedoch vom Server beim Verarbeitungsprozess berücksichtigt werden.

3.4.3 Exploit

Im letzten Schritt werden die gesammelten Informationen zusammengetragen und ein Angriff wird konstruiert.

Ein Angriff ist kombinierbar mit anderen bekannten Angriffsmethoden. Dies hängt immer davon ab, was genau die Sicherheitslücke zulässt. Einer der verheerendsten Angriffe ist die Kombination mit Cross-Site-Scripting (XSS). Mithilfe des Caches wird aus dem Web Cache Poisoning Angriff ein stored XSS Angriff auf alle Benutzer*innen die einen bestimmten Endpunkt aufsuchen.

3.4.4 Cachebuster

Ein Cachebuster wird verwendet, um den Caching-Mechanismus zu behindern. Das Hinzufügen eines eindeutigen Erkennungsmerkmals verhindert, dass der Cache eine Response generieren kann. Wie in Abschnitt 3.2 beschrieben, wird für jede Request anhand einer Anzahl vordefinierter Header ein Cache-Key erzeugt. Der Cachebuster soll verhindern, dass zukünftige Requests den gleichen Cache-Key besitzen. Daraus resultiert, dass der Cache keinen passenden Cache-Key findet und

folglich für jede Request, die mit einem Cachebuster versehen ist, einen neuen Eintrag generiert. Diese bedeutet wiederum, dass jede Request vom Server beantwortet wird.

Code 3.3 zeigt ein einfaches Beispiel, wie ein Cachebuster angewendet werden kann. Dabei wird dem Query-String ein neuer Parameter angehängt, der als Cachebuster dient.

```
1 GET /index.php?q=someParam&cachebuster=1293102931123981029 HTTP/1.1
2 Host: example.com
3 ...
```

CODE 3.3: Cachebuster im Einsatz.

Ist der Query-String nicht Teil des Cache-Keys, dann erhält man die Response vom Cache, da `GET /` und `GET /?cachebuster=1293102931123981029` gleich aussehen für den Cache. Ist der Query-String jedoch Teil des Cache-Keys, wird ein neuer Eintrag generiert. Folglich darf beim Testen von weiteren Requests dieser Cachebuster nicht mehr verwendet werden, da sonst der Cache die Response generieren würde und nicht der Server.

Wenn der Query-String nicht Teil des Cache-Keys ist, gibt es dennoch Möglichkeiten, den Cache auszutricksen. Jeder Header, der im Cache-Key aufgenommen wird, bietet eine mögliche Einstiegsstelle für einen Cachebuster. Es muss jedoch darauf geachtet werden, dass der Cachebuster nicht vom Server im Verarbeitungsprozess mitbeachtet wird, was die Response verfälschen könnte.

Code 3.4 zeigt mögliche Einstiegsstellen für einen Cachebuster, falls dieser nicht als Parameter eingesetzt werden kann. Wie bereits erwähnt: Damit der Cachebuster funktioniert, muss der Header im Cache-Key vorhanden sein.

```
1 GET /?CachebusterIsUselessHere=1234 HTTP/1.1
2 Host: example.com
3 Accept-Encoding: gzip, compress, cachebuster1234
4 Accept: */*, text/cachebuster1234
5 Cookie: cachebuster1234=1
6 Origin: https://cachebuster1234.example.com
```

CODE 3.4: Erweitertes Beispiel, wie der Cachebuster verwendet werden kann.

3.5 Gegenmassnahmen

Die sicherste Gegenmassnahme ist es, keinen Cache zu verwenden [11]. Dies mag jedoch keine Option für die Mehrheit aller Webseiten sein. Falls es jedoch eine Option sein könnte, sollte genau evaluiert werden, ob der Cache wirklich benötigt wird.

Wenn ein Cache verwendet wird, sollen die folgenden Punkte beachtet werden:

- **Vertraue keinen HTTP Headers [13]:** Headers, die nicht im Cache-Key verwendet werden, sollen nicht vom Server benutzt werden. Die Verwendung von unkeyed Headers kann dazu führen, dass die Applikation anfällig für Web Cache Poisoning wird.
- **Kenne die Frameworks der Drittanbieter:** Viele Systeme basieren auf der Hilfe von anderen Systemen. Daher ist es wichtig zu wissen, wie diese Sicherheitsfeatures verwendet werden müssen.

- **Vertraue keinen GET-Request Bodies [13]:** GET-Requests sollten eigentlich keinen Body besitzen, sie können es aber [14]. Aus Sicherheitsgründen sollten GET-Requst Bodies nicht vertraut werden.

4 Anforderungen

Dieses Kapitel dient der Dokumentation aller Anforderungen, die für die CTF Challenges gelten. Die Anforderungen werden nach Kategorien gruppiert und mit einer eindeutigen ID versehen, um eine lückenlose Rückverfolgbarkeit während des gesamten Entwicklungsprozesses sicherzustellen.

4.1 Nichtfunktionale Anforderungen

Tabelle 4.1 listet alle nichtfunktionalen Anforderungen auf, welche an die CTF Challenges gestellt wurden.

TABELLE 4.1: Nichtfunktionale Anforderungen der CTF Challenges.

ID	Anforderung
Req-NA-01	Die CTF Challenges werden für Spieler*innen gestaltet, die bereits Erfahrung mit CTFs gemacht haben.
Req-NA-02	Die CTF Challenges Easy und Medium setzen voraus, dass ein Grundwissen über einfache Angriffe im Bereich XSS vorhanden ist.
Req-NA-03	Die CTF Challenges dienen zur Wissensübertragung/-vermittlung. Eine Challenge soll dabei Spielenden neues Wissen beibringen oder ihr Wissen herausfordern, um dieses zu festigen oder zu erweitern.
Req-NA-04	Die CTF Challenges sollen mit Humor und kleinen Gimmicks unterhalten. Die Art der Unterhaltung kann dabei in Form von Text oder Bild stattfinden.
Req-NA-05	Hints für die CTF Challenges Easy und Medium sollen zu keinem Punkteabzug führen.
Req-NA-06	Die Challenges sollen für mindestens 60 Personen gleichzeitig spielbar sein.
Req-NA-07	Die Challenges sollen realitätsnahe sein, damit aus dem gewonnen Wissen Profit geschlagen werden kann.
Req-NA-08	Die CTF Challenges Hard soll auf einer realen Sicherheitslücke aufbauen.
Req-NA-09	Hints für die CTF Challenge Hard sollen zu einem Punkteabzug führen.

Req-NA-02 greift bereits vor auf die spezifischen Challenges (siehe Kapitel 5). Wie bereit in Kapitel 3.3 beschrieben, ist Web Cache Poisoning eine Einstiegsmöglichkeit, um weitere Sicherheitslücken auszunutzen. Eine häufig auftretende Schwachstelle, die durch Web Cache Poisoning ausgenutzt werden kann, ist XSS. Aus diesem Grund wurde entschieden, dass die Challenges ebenfalls von dieser Schwachstelle Gebrauch machen.

4.2 Technische Anforderungen

Tabelle 4.2 listet alle technischen Anforderungen auf, welche an die CTF Challenges gestellt wurden.

TABELLE 4.2: Technische Anforderungen an die CTF-Challenges.

ID	Anforderung
Req-TA-01	Es soll mehrere Schwierigkeitsstufen zum Thema Web Cache Poisoning geben.
Req-TA-02	Die Challenges sollen die Benutzung von Param Miner [15] (oder ähnliche Tools für Brute-Force-Angriffe) zulassen bzw. standhalten.
Req-TA-03	Eine Challenge muss gleichzeitig von mehreren Personen genutzt werden können. D. h. ein Angriff von Person A darf sich nicht auf die Challenge von Personen B, C usw. auswirken.
Req-TA-04	Es soll nur die beabsichtigte Sicherheitslücke für eine Challenge ausgenutzt werden können, um das Flag zu erhalten.
Req-TA-05	Alle Challenges sollen die gleiche Struktur zugrunde haben.
Req-TA-06	Das System soll offen für neue Challenges (neue Angriffsmuster) sein.

Req-TA-01 besagt, dass es mehrere Schwierigkeitsstufen geben soll. Da diese Anforderung noch viel Interpretationsspielraum offen hält, wurde entschieden, dass es drei Challenges geben soll. Die nachfolgenden drei Tabellen beschreiben die Anforderungen an die Challenge Easy (siehe Tabelle 4.3), die Challenge Medium (siehe Tabelle 4.4) und die Challenge Hard (siehe Tabelle 4.5).

TABELLE 4.3: Spezifizierung der Anforderung Req-TA-01 für die Challenge Easy.

ID	Anforderung
Req-TA-01.Easy.1	Die Challenge ist ein Tutorial. Dem*Der Spieler*in soll Schritt für Schritt das Prinzip von Web Cache Poisoning erklärt werden.

TABELLE 4.4: Spezifizierung der Anforderung Req-TA-01 für die Challenge Medium.

Id	Anforderung
Req-TA-01.Medium.1	Die Challenge soll auf dem Tutorial (Challenge Easy) basieren. Der*Die Spieler*in soll dabei erlerntes Wissen festigen können.
Req-TA-01.Medium.2	Die Challenge soll die Denkweise eines Angreifers (siehe 3.4) trainieren.

TABELLE 4.5: Spezifizierung der Anforderung Req-TA-01 für die Challenge Hard.

ID	Anforderung
Req-TA-01.Hard.1	Die Challenge soll den*die Spieler*in vor eine Herausforderung stellen.
Req-TA-01.Hard.2	Die Challenge soll auf einer realen Sicherheitslücke basieren.

In Req-TA-03 sind noch ein paar technische Details enthalten, welche auf den ersten Blick nicht ersichtlich sind. Bei Web Cache Poisoning ist nicht die Applikation das primäre Angriffsziel, sondern der zuvorliegende Cache (siehe Kapitel 3). Aus diesem Grund ist es wichtig, dass jeder*jede Spieler*in seine*ihre eigen Umgebung besitzt, mit welcher er*sie interagieren kann.

4.3 ZHAW Umgebung

Wie bereits in der Einleitung und in der Aufgabenstellung erwähnt (siehe Kapitel 1 und Abschnitt 1.4) besitzt die ZHAW eine CTF Plattform. Tabelle 4.6 fasst alle Anforderungen zusammen, die von den CTF Challenges beachtet werden müssen, damit sie in die ZHAW Umgebung integriert werden können.

TABELLE 4.6: Anforderungen zur Interoperabilität der Challenges mit der ZHAW Umgebung.

ID	Anforderung
Req-IOP-01	Die Erstellung und Zuweisung von individuellen Challenges soll von einem zentralen System gesteuert werden.
Req-IOP-02	Die Punktvergabe soll von der ZHAW-Plattform verwaltet werden.
Req-IOP-03	Hints für eine Challenge sollen auf der ZHAW-Plattform ersichtlich sein.

Req-IOP-01 erwähnt ein zentrales System. Mit CTFd [1] ist es möglich, auf externe Challenges zu verlinken. Um von CTFd auf eine Challenge zu verlinken, muss der Ort (URL) bekannt sein. Die Challenges befinden sich jedoch in ihrer eigenen Umgebung (siehe Req-TA-03). Damit nun auf eine Challenge verwiesen werden kann, muss ein zentrales System entwickelt werden, welches die Verwaltung der Umgebungen für die verschiedenen Challenges übernimmt. Dieses System kann dann an die ZHAW-Plattform angebunden werden.

4.4 Sicherheit

Die Challenges sollen sicher sein. D. h. eine Challenge soll von Fremdeinwirkungen geschützt werden. Tabelle 4.7 beschreibt die Anforderungen, die der Sicherheit der Challenges dienen soll.

TABELLE 4.7: Anforderungen an die Sicherung der Challenges.

ID	Anforderung
Req-S-01	Eine Challenge soll nur von dem*der Spieler*in gespielt werden können, der*die Zugriff auf die Challenge besitzt.
Req-S-02	Ein*e Spieler*in soll nicht unendlich viele Umgebungen erzeugen können.

Die Anforderungen beziehen sich stark auf das zentrale System, welches in Req-IOP-01 erwähnt wurde. Wie in Req-S-01 bereits erwähnt, soll eine Fremdeinwirkung von anderen Spieler*innen verhindert werden, da diese sonst den Cache mit Anfragen überfluten könnten, was zur Folge hat, dass die Challenge nicht mehr spielbar

ist. Req-S-02 befasst sich ebenfalls mit der gleichen Problematik. Ein*e Spieler*in soll nicht unendlich viele Umgebungen erzeugen können und somit anderen Spieler*innen den Zugriff auf eine Challenge blockieren.

5 Konzept

Bevor mit der Umsetzung begonnen werden konnte, wurde ein Konzept ausgearbeitet. Dabei wurde erörtert, wie die Sicherheitslücke der jeweiligen Challenge aussehen soll. Ausserdem wurde ein Setting erstellt, in welchem sich die Challenges befinden.

Nachfolgend werden die drei geplanten Challenges genauer erläutert und es wird die Webseite vorgestellt, welche bei allen Challenges als Basis dient.

5.1 Challenges

Es wurden drei Challenges geplant. Die Schwierigkeit der Challenge reicht dabei von einfach bis schwer (siehe Req-TA-01).

Angehend werden die drei Challenges genauer vorgestellt. Dabei wird für jede Challenge die folgenden Fragen beantwortet:

- **Ziel** - Was ist das Ziel der Challenge?
- **Schwierigkeit** - Welche Skills werden benötigt, um die Challenge erfolgreich zu meistern?
- **Sicherheitslücke** - Wie sieht die eingebaute Sicherheitslücke aus?
- **Ablauf** - Wie sieht der Ablauf der Challenge aus? D. h. welche Schritte müssen unternommen werden, um an das Flag zu gelangen?
- **Hints** - Welche Hinweise werden den Spieler*innen zur Verfügung gegeben?

5.1.1 Easy: How to Web Cache Poisoning

Ziel: Die Challenge Easy fungiert als Tutorial und bietet eine Einführung in das Thema. Dem*Der Spieler*in soll das nötige Wissen über Web Cache Poisoning vermittelt werden. Dabei wird er*sie einen unkeyed Header ausnutzen, um eine JS-Injection durchzuführen.

Schwierigkeit: Um das Tutorial zu spielen, wird kein Wissen im Bereich Web Cache Poisoning benötigt. Das Tutorial soll dem*der Spieler*in dieses Wissen übermitteln. Jedoch setzt die Challenge voraus, dass der*die Spieler*in bereits Erfahrung mit Angriffen im Bereich Websicherheit mit bringt.

Sicherheitslücke: Die Sicherheitslücke setzt sich aus einem selbst erstellten Header, welcher nicht mehr gebraucht, aber von der RestAPI immer noch unterstützt wird und einer JS-Injection zusammen.

Dem*Der Spieler*in stehen folgende Ressourcen zur Verfügung:

> **Cache:** Server akzeptiert den `x-forwarded-host` Header, der jedoch nicht im Cache-Key vorhanden ist.

JS-Injection: `x-forwarded-host` kann verwendet werden, um die URL im Pfad für ein Bild zu spezifizieren, das angezeigt werden soll. Dieser Pfad wird von der Webseite auf unsichere Art und Weise verwendet.

Ablauf: Mithilfe einer Schritt-für-Schritt Anleitung wird der*die Spieler*in durch die Challenge geführt. Der Ablauf ist wie folgt:

1. Die Request für die Detail-Seite muss abgefangen werden.
2. Die gefangene Request wird manipuliert, indem eine JS-Injection (siehe Code-Ausschnitt 5.1), in den `X-Forwarded-Host`-Header eingefügt wird.

```
1 GET /lessons/4? HTTP/1.1
2 X-Forwarded-Host: #" onerror=alert('FLAG{ValueOfTheFlag}')>
```

CODE 5.1: Infizieren des Caches der Easy Challenge.

3. Nach der Manipulierung muss die abgefangene Request an den Server weitergeleitet werden.
4. Danach wird die Request vom Server bearbeitet. Sofern die Seite nicht bereits im Cache vorhanden ist, wird die Request gecached.
5. Nun kann die Detail-Seite in einem anderen Browser geöffnet werden. Dabei wird der eingefügte «alert()», und somit auch das Flag, auf der Webseite dargestellt.

Da es ein Tutorial ist, wird dem*der Spieler*in das Flag bereits zu Beginn der Challenge zur Verfügung stehen.

Hints: Es gibt keine Hints, da die Schritt-für-Schritt Anleitung alle benötigten Informationen beinhaltet.

5.1.2 Medium: Cross Site Cookie Mining

Ziel: Der*Die Spieler*in soll sein*ihr Können unter Beweis stellen und zeigen, dass er*sie ein Session-Token eines Admins stehlen kann. Dabei wird ein vergessener Header ausgenutzt, über diesen es möglich ist, einen XSS-Angriff durchzuführen.

Schwierigkeit: Das Vorgehen ist identisch wie dasjenige des Tutorials. Zuerst soll der unkeyed Header entdeckt werden, welcher vom Server bei der Request-Verarbeitung mitbeachtet wird und danach muss ein XSS-Angriff konstruiert werden. Das Nichtvorhandensein der Anleitung soll den*die Spieler*in vor eine Herausforderung stellen. Dabei kann er*sie die Methodik eines Web Cache Poisoning Angriffs trainieren (siehe 3.4).

Sicherheitslücke: Die Sicherheitslücke setzt sich aus einem selbst erstellten Header, welcher nicht mehr gebraucht, aber von der RestAPI immer noch unterstützt wird und einem XSS-Angriff zusammen.

Dem*Der Spieler*in stehen folgende Ressourcen zur Verfügung:

Cache: Server akzeptiert den `x-images` Header, der jedoch nicht im Cache-Key vorhanden ist.

XSS: `x-images` kann verwendet werden, um den Pfad für ein Bild, das angezeigt werden soll, zu spezifizieren. Dieser Pfad wird von der Webseite auf unsichere Art und Weise verwendet.

Catcher: Der Catcher ist ein Service, der benutzt werden kann. Er dient als Auffangnetz von Requests. D. h. alle Requests die an den Catcher gesendet werden, werden auf einer Webseite dargestellt. Er dient als Hilfsmittel zum Lösen der Challenge.

Ablauf: Ein Bot schaut sich regelmässig Einträge auf der Webseite an. Um dessen Token zu erhalten, soll wie folgt vorgegangen werden:

1. Sich auf der Webseite umschauen und ein paar Requests absenden.
2. Diese Requests manipulieren und schauen, wie sich die Webseite verhält.
3. Damit man die Detail-Seiten betrachten kann, muss man sich zuerst anmelden, das Login erhält man aus der angezeigten Fehlermeldung auf der Detail-Seite.
4. Falls der versteckte unkeyed Header noch nicht gefunden werden konnte, den Param-Miner der Burp-Suite einsetzen, dieser sollte den `x-images`-Header finden.
5. Sobald der unkeyed Header gefunden wurde, kann eine XSS-Attacke ausgeführt werden, siehe Code-Ausschnitt 5.2.

```
1 GET /lessons/4? HTTP/1.1
2 x-images: #" onerror="XSSImage=new Image; XSSImage.src='catcherUrl
    /requests/test?cookie='+document.cookie;" />
```

CODE 5.2: Infizieren des Caches der Medium Challenge.

6. Der injizierte Code liest beim Ausführen das Token des Benutzers aus und sendet dieses an einen Catcher.
7. Der Catcher listet alle empfangen Tokens auf und die Challenge ist gelöst.

Das Token ist das gesuchte Flag, welches zum Erhalten der Punkte angegeben werden muss.

Hints: Falls ein*e Spieler*in an seine Grenzen kommt, stehen vier Hints für ihn*sie zur Verfügung. Die Hints sind folgende:

1. «Param Mining wurde im Tutorial erwähnt, vielleicht lohnt es sich das nochmals anzuschauen.»
   Dieser Hint soll die Spielenden auf Param Mining aufmerksam machen.
2. «Vielleicht kann XSS mit Cache Poisoning kombiniert werden.»
   Dieser Hint soll die Spielenden dazu führen, XSS mit Cache Poisoning zusammen einzusetzen.
3. «Cookies beinhalten manchmal mehr Informationen, wie sie sollten.»
   Dieser Hint soll darauf hinweisen, dass im Cookie etwas zu finden sein wird.
4. «Catcher-Seiten können hilfreich dabei sein, Informationen von anderen Benutzenden zu erhalten.»
   Dieser Hint soll aufzeigen, dass der Catcher für das Fangen der Cookies verwendet werden kann.

5.1.3 Hard: Double Cache Mayhem

Ziel: Der*Die Spieler*in soll aktiv herausgefordert werden. Ziel der Challenge ist es ein MITM-Angriff auf das Login-Formular der Webseite durchzuführen, um Benutzername und Passwort eines Bots abzufangen.

Schwierigkeit: Um diese Challenge zu lösen, wird ein tieferes Verständnis der Verhaltensweise eines resp. mehreren Caches benötigt. Ausserdem sollten bekannte Headers und ihr Verwendungszweck bekannt sein.

Sicherheitslücke:[8][10][9] Die Sicherheitslücke setzt sich aus mehreren Teilen zusammen.

Dem*Der Spieler*in stehen folgende Ressourcen zur Verfügung:

> **RestAPI:** Die RestAPI akzeptiert einen Parameter `destination=value`. Wenn dieser gesetzt wird, dann ist der Wert in der Response als `Location: value` vorhanden.
>
> **Interner Cache:** Beim internen Cache lassen sich die Werte eines Cache-Keys überschreiben. Wird in einer Request der Header `X-Original-URL` mitgesendet, dann wird dieser vom Cache und Backend priorisiert. D.h. dieser wird verwendet und die URL wird ignoriert. Parameter, welche in der URL sind, werden aber immer noch vom Server bei der Verarbeitung einer Request beachtet.
>
> **Externer Cache:** Der externe Cache besitzt keine Sicherheitslücke, ist jedoch eine Hürde, die überwunden werden muss.
>
> **Catcher:** Der Catcher ist ein Service, der benutzt werden kann. Er dient als als Auffangnetz von Requests. D. h. alle Requests, die an den Catcher gesendet werden, werden auf einer Webseite dargestellt. Er ist ein Hilfsmittel zum Lösen der Challenge.
>
> **MITM-Webseite:** Sie is eine Mock-Webseite, die zum Abfangen des Logins dient. Alle Loginversuche werden by default an den Catcher gesendet. Sie ist ebenfalls ein Hilfsmittel zum Lösen der Challenge.

Ablauf: Ein Bot loggt sich regelmässig auf der Login-Seite ein. Um dessen Benutzername/Passwort zu erhalten, soll wie folgt vorgegangen werden:

1. Eine Request absenden bei welcher `?destination=http://mitm-webseite.com` als Parameter vorhanden ist. In der Response wird `Location: http://mitm-webseite.com` zu sehen sein.
2. Eine Request suchen, die in der Response einen Redirect enthält. Diese Request (bspw. `/login`) kann manipuliert werden, dass neu auf eine böse Seite weitergeleitet wird.
3. Zuerst muss der interne Cache die neue Location zurückgeben. Dazu wird der Pfad `/attack` infiziert (siehe 5.3). Der Cache speichert einen Eintrag auf die URL `/attack`.

```
1 GET /?destination=https://mitm-webseite.com HTTP/1.1
2 Host: example.com
3 X-Original-URL: /attack
```

CODE 5.3: Infizieren des internen Caches.

4. Danach wird der externe Cache mit der bösartigen Response beladen. Dabei wird der Pfad `/login` infiziert (siehe 5.4). Wenn die Request bei der RestAPI ankommt, schaut der interne Cache, ob er eine Antwort für `/attack` besitzt, welche er dann als Response zurückgibt.

```
GET /login HTTP/1.1
Host: example.com
X-Original-URL: /attack
```

CODE 5.4: Infizieren des externen Caches.

5. Überprüfen, dass eine Weiterleitung auf die bösartige Webseite geschieht. Der Bot loggt sich auf der Seite ein und man erhält Benutzername/Passwort, welche das Flag beinhalten.

Hints:

1. «Es könnte sich lohnen das Login genauer anzuschauen.»
   Dieser Hint soll die Spielenden darauf hinweisen, dass das Login die Sicherheitslücke enthält.
2. «Mehrere Caches könnten diesmal eingesetzt worden sein.»
   Dieser Hint zeigt den Spielenden, dass mehrere Caches verwendet werden.
3. «Mehrere Schritte sind nötig.»
   Dieser Hint zeigt den Spielenden, dass die Lösung diesmal nicht so einfach wird, wie bei den vorherigen Challenges und mehrere Schritte durchgeführt werden müssen.
4. «CVE-2018-14773»
   Bei diesem Hint wird den Spielenden für den Einsatz von 250 Punkten die CVE, also die Sicherheitslücke, vorgegeben. Damit sollte die Lösung der Challenge erleichtert werden.

5.2 Webseite

In diesem Kapitel wird der Aufbau und die Planung der Webseite erläutert, damit in den folgenden Challenge-Kapiteln klar ist, wie diese Challenges aussehen und wieso diese so aufgebaut wurden.

5.2.1 Aufbau der Webseite

Damit die Challenges möglichst realitätsnah und auch anschaulich sind, wurde eine Webseite erstellt. Diese dient dazu, dass die Spielenden eine normale Plattform vor sich haben, wo sich verschieden Schwachstellen eingeschlichen haben. Um die Challenge-Webseite zu erstellen, musste zuerst ein Thema/Zweck der Webseite ermittelt werden. Danach wurden Mockups der Webseite erstellt, damit man sich diese vorstellen und diese auch zuerst ausarbeiten kann, bevor man die eigentliche Webseite implementieren muss. Das ausgewählte Thema ist ein ZHAW-Coaching-Portal, wo Tutoren eine Plattform haben, um ihre Dienste (Nachhilfestunden etc.) anzubieten. Die Webseite ist so aufgebaut, dass es eine Homepage hat (siehe Abbildung 5.1), auf welcher alle verfügbaren Tutoren aufgelistet werden.

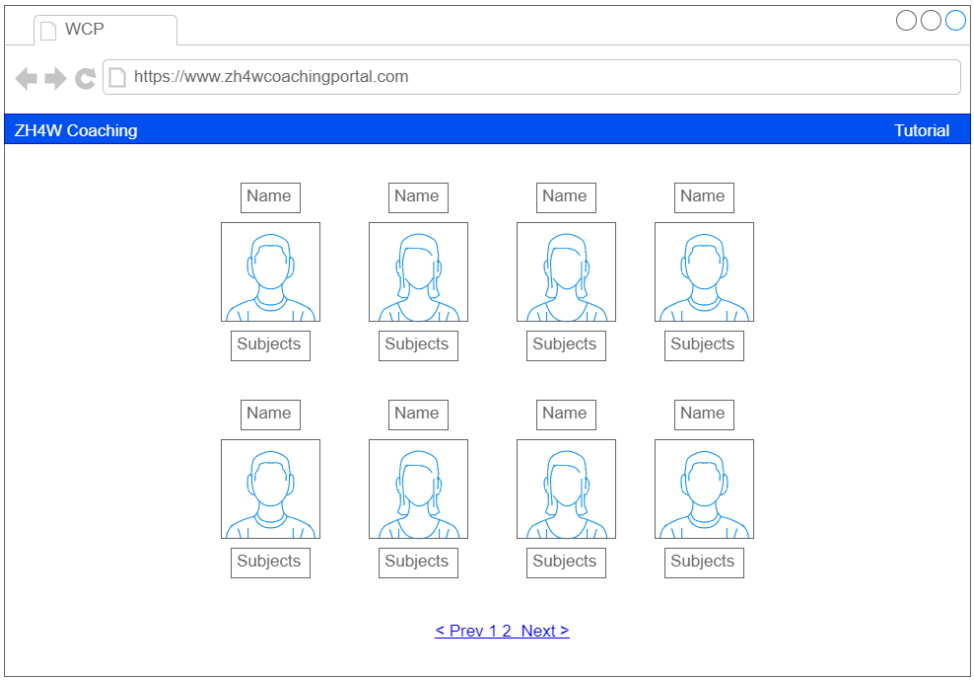


ABBILDUNG 5.1: Mockup der Homepage.

Dies dient dazu, dass sich die Studierenden wie in einem Online-Shop einen*eine Tutor*Tutorin auswählen können. Um genauere Informationen über einen*eine Tutor*Tutorin zu erhalten kann er*sie in der Homepage ausgewählt werden und die Detail-Seite, in Abbildung 5.2 zu sehen, öffnet sich. Auf dieser Detail-Seite können dann alle wichtigen Informationen entnommen werden.

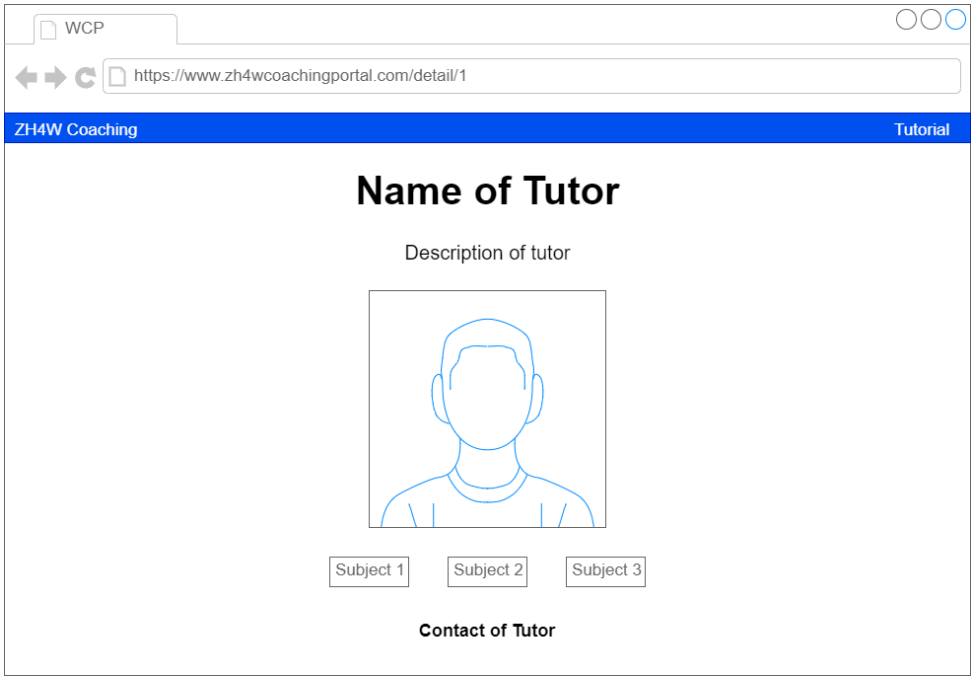


ABBILDUNG 5.2: Mockup der Detail-Ansicht.

Für die Easy Challenge ist zusätzlich zur normalen Verwendung der Webseite noch

eine Tutorial-Seite (siehe Abbildung 5.3) geplant, wo Web Cache Poisoning kurz erklärt und die notwendigen Schritte für die Lösung der Challenge aufgezeigt werden.

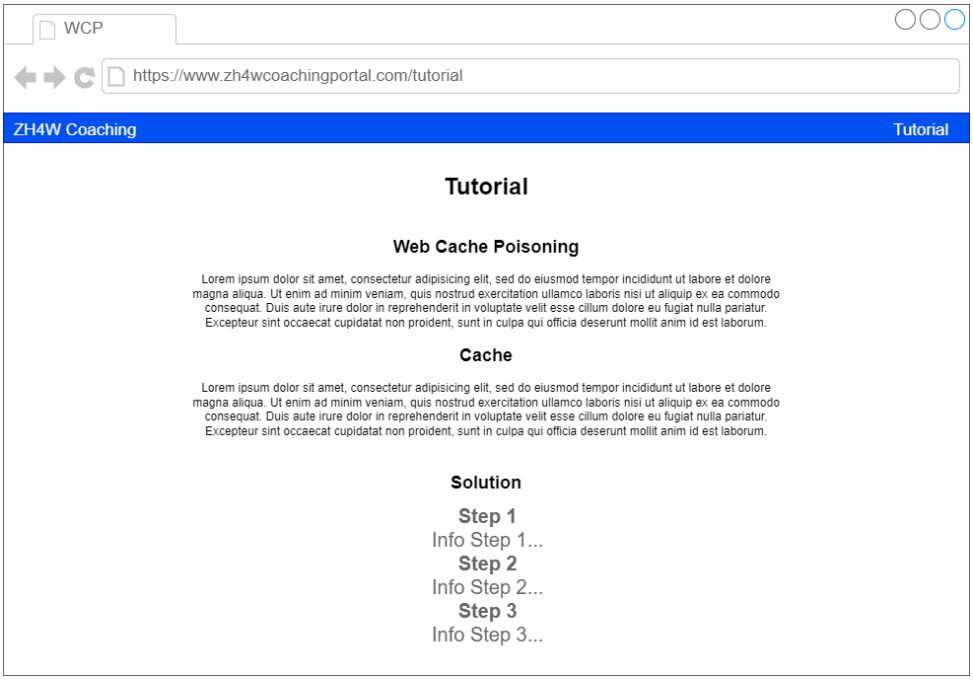


ABBILDUNG 5.3: Mockup der Tutorial-Seite.

Für die Medium- und Hard-Challenge wird die Tutorial-Seite durch eine Login-Seite ersetzt. Für diese wurden keine Mockups erstellt, da eine einfache Login-Form mit zwei Input-Feldern (Email und Passwort) sowie einem Submit- und Cancel-Button geplant wurde.

5.2.2 Entscheidungsfaktoren

Diese Art von Webseite wurde ausgewählt, da mit der Verlinkung zu einer Detail-Seite eine einfache Einbindung einer Schwachstelle möglich wird. Denn sobald eine Detail-Seite aufgerufen wird, muss das Frontend eine neue Request absetzen, welche dann vom Server gecached werden kann. Ebenfalls wurde beachtet, dass die Webseite einfach anzupassen ist (Beispiel Tutorial durch Login ersetzen), da drei Challenges darauf basieren und in Zukunft auch noch weitere dazukommen können.

Um die Webseite möglichst realitätsnah zu gestalten, wurden auch Bilder von Personen verwendet, welche die Tutoren darstellen sollen. Damit es keine rechtlichen Probleme gibt, wurden AI-Generated-Images verwendet [16]. Dies gewährleistet, dass keine echten Personen verwendet werden.

6 Architektur

Dieses Kapitel befasst sich mit der umgesetzten Architektur. Einerseits wird beschrieben, wie die Challenges umgesetzt wurden und ausserdem wird das zentrale System (Req-IOP-01) erklärt, welches sich über drei Server erstreckt.

Der Aufbau des Kapitels ist wie folgt gestaltet:

- **Verwendete Technologien**

  Erläutert die Technologie, die für die Umsetzung der verschiedenen Komponenten verwendet wurde.

- **Integration in die ZHAW CTFd Infrastruktur**

  Beschreibt das zentrale System, welches entwickelt wurde, um die Challenges in die CTFd Umgebung der ZHAW einzubinden.

- **Sicherung der Challenges vor Fremdeinwirkung**

  Legt dar, wie die Challenges vor Fremdeinwirkungen geschützt werden.

- **Challenges**

  Illustriert die Umsetzung der drei Challenges. Dabei wird erläutert wie die Challenges allgemein aufgebaut sind. Was spezifisch für eine jeweilige Challenge ist und wie sie containerisiert werden.

- **Testkonzept**

  Die einzelnen Teststufen werden in diesem Abschnitt erläutert. Was wird getestet - und wieso.

6.1 Verwendete Technologien

Bei der Technologie wurde auf JavaScript gesetzt. Bei der Webseite kamen lediglich noch HTML und CSS dazu. Es wurde sich gegen die Verwendung eines Frameworks entschieden, um möglichst viel Kontrolle zu besitzen. Für die Erstellung der verschiedenen Server griff man auf das Express.js-Framework [17] zu. Falls serverseitig Webseiten generiert werden sollten, kam die Template-Engine Pug.js [18] zum Einsatz.

Für das Testing wurde Mocha [19] i.V.m Sinon [20] verwendet. Diese kamen hauptsächlich im Backend zum Einsatz, um die Anwendungslogik zu testen.

Für die Containerisierung wurde Docker und Docker Compose gebraucht. Die Server C2-Server (C2: Command and Controll), CAS-Server (CAS: Central Authentication Service) und Challenge Manager besitzen alle ein eigene `docker-compose.yml` Datei, welche ein einfaches Starten und Stoppen des jeweiligen Servers ermöglicht.

Für die Challenges wurden individuelle Dockerfiles erstellt, was eine einfache Verwaltung der Container-Umgebung ermöglicht, da alle benötigten Ressourcen in nur einem Container sind.

Nginx wird als Webserver verwendet und bei Bedarf auch als Reverse-Proxy.

Für den Bot (siehe Abschnitt 5.1.2 und Abschnitt 5.1.3), welcher zur Verifizierung der Challenges dienen soll, wurde Selenium [21] i.V.m Python verwendet.

6.2 Integration in die ZHAW CTFd Infrastruktur

Damit ein*e Spieler*in eine Challenge spielen kann, muss zuerst eine virtuelle Umgebung erzeugt werden. Das Erzeugen dieser Umgebungen wird vom Challenge Manager und dem C2-Server gehandhabt.

Der **Challenge Manager** dient als Bindeglied zwischen der ZHAW-Plattform und des C2-Servers. Über ein Interface ist es möglich, eine neue Challenge zu starten. Dabei kümmert sich der Challenge Manager um das ganze Setup. Das Resultat ist eine Webseite, auf welcher alle Informationen vorhanden sind, welche zum Spielen besagter Challenge nötig werden.

Der **C2-Server** bildet die Verwaltungseinheit aller Challenges. Auf Anfrage kann eine neue Container-Umgebung erstellt werden. Der C2-Server kümmert sich dabei um den gesamten Lebenszyklus eines Containers. Die Lebenszeit eines Containers ist dabei fix vordefiniert. Es ist daher nur möglich eine Umgebung anzufordern, jedoch nicht, sie wieder zu beenden/entfernen.

Abbildung 6.1 zeigt, wie Challenge Manager und C2-Server an die ZHAW Infrastruktur angebunden werden.

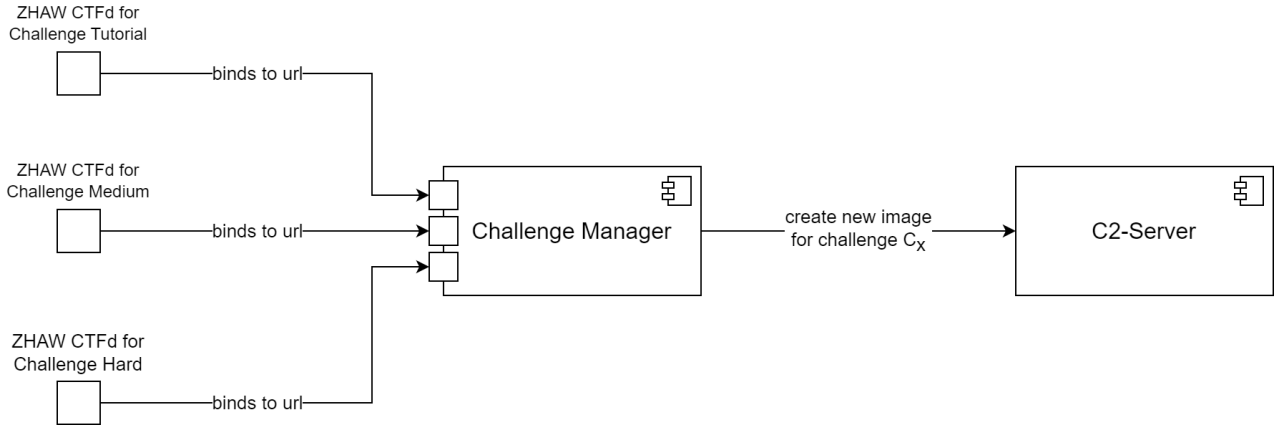


ABBILDUNG 6.1: Einbindung des Challenge Manager und C2-Servers in die ZHAW CTFd Infrastruktur.

Erkennbar ist, dass von ZHAW CTFd drei Pfeile auf jeweils einen Ports des Challenge Managers zugreifen. Diese stellt die Verlinkung dar. Klickt jemand auf den Link bei ZHAW CTFd, dann wird dem Challenge Manager signalisiert, dass eine neue Umgebung erstellt werden soll. Abbildung 6.2 veranschaulicht diesen Ablauf. Nachdem die Umgebung für die Challenge erstellt wurde (Schritt: return container ports), wird vom Challenge Manager eine neue Seite erstellt, die alle Informationen für die Challenge besitzt (Abbildung 6.3).

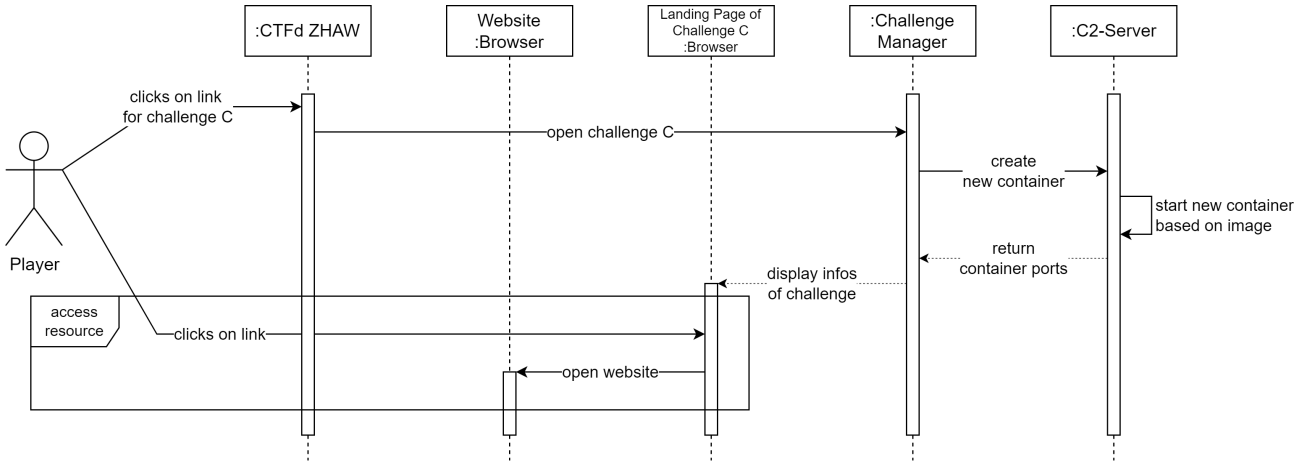


ABBILDUNG 6.2: Ablauf beim Zugreifen auf eine Challenge.

Abbildung 6.3 zeigt ein Mockup der Landing Page der Challenge Medium. Die Gestaltung dieser Seiten ist auf das Nötigste beschränkt. Sie besitz einen Titel, eine kurze Einleitung zur Challenge, die Links zu den Ressourcen und eine kurze Erläuterung der Ressourcen. Klickt der*die Spieler*in nun auf einen Link (Prozess: access resource), wird die entsprechende Resource in einem eigenen Browserfenster geöffnet.

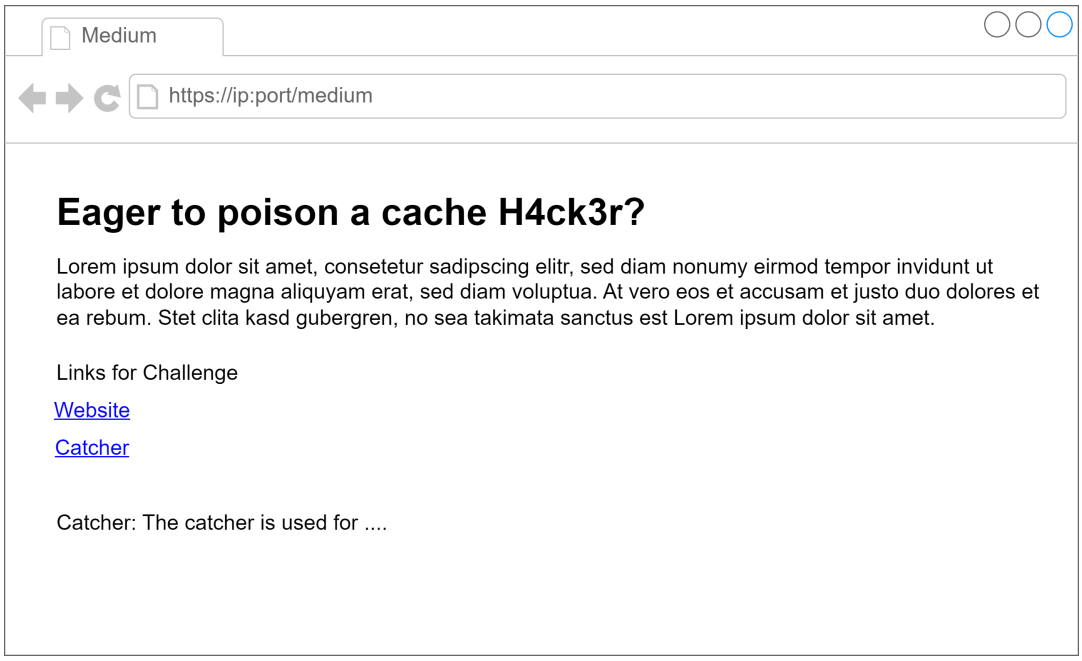


ABBILDUNG 6.3: Mockup der Landing Page des Challenge Managers für die Challenge Medium.

6.3 Sicherung der Challenges vor Fremdeinwirkung

Eine Anforderung an die Challenges ist es, dass ein*e Spieler*in nur Zugriff auf seine*ihre Challenges besitzt (siehe Req-S-01). Da das CTFd-Framework keine Möglichkeit besitzt, einen Token für einen*eine Spieler*in oder ein ganzes Team zu generieren [1], wurde ein CAS-Server entwickelt. Abbildung 6.4 zeigt, wie der CAS-Server mit dem C2-Server und einer Challenge interagiert.

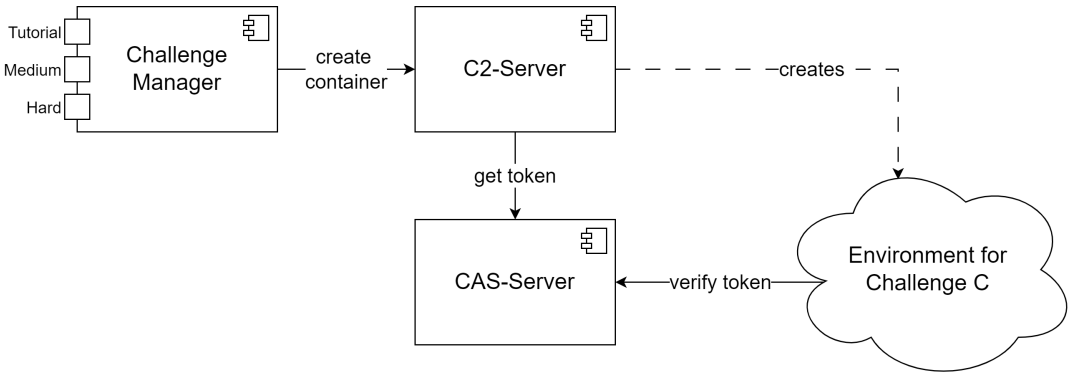


ABBILDUNG 6.4: Einbindung des CAS-Servers.

Wenn der C2-Server eine neue Challenge-Umgebung erstellt, wird er zuvor einen Token vom CAS-Server verlangen. Dieser Token dient zur Identifizierung, ob jemand Zugriff auf die RestAPI besitzt oder nicht. Dies geschieht, in dem der Challenge Manager den Token als Cookie im Browser speichert. Öffnet der*die Spieler*in anschliessend die Webseite der Challenge, wird der Token aus den Cookies ausgelesen und bei jeder Request mitgesendet. Abbildung 6.5 zeigt, wie der Token die RestAPI und damit den Cache vor Fremdeinwirkung schützt. Da der Token bei jeder Request an die RestAPI gesendet wird, braucht auch der Bot diesen Token. Der Token wird aus diesem Grund auch vom C2-Server über eine Umgebungsvariable an den Container der Challenge übergeben.

Durch den Token kann der Cache nun von DOS-Angriffen geschützt werden. Ist der Token nicht vorhanden oder ungültig, wird die Anfrage nicht weiter bearbeitet, was zur Folge hat, dass die Anfrage den Cache nie erreicht.

Die zusätzlich gewonnene Sicherheit besitzt aber auch einen Nachteil. Da das Cookie nur in dem Browser verfügbar ist, über welchen mit dem Challenge Manager die Challenge erstellt wurde, muss das Cookie in allen anderen Browsern manuell importiert werden. Ansonsten ist die Challenge nicht spielbar. Dies gilt besonders für Tools wie bspw. Burp Suite, welche dazu dienen, Requests/Responses abzufangen.

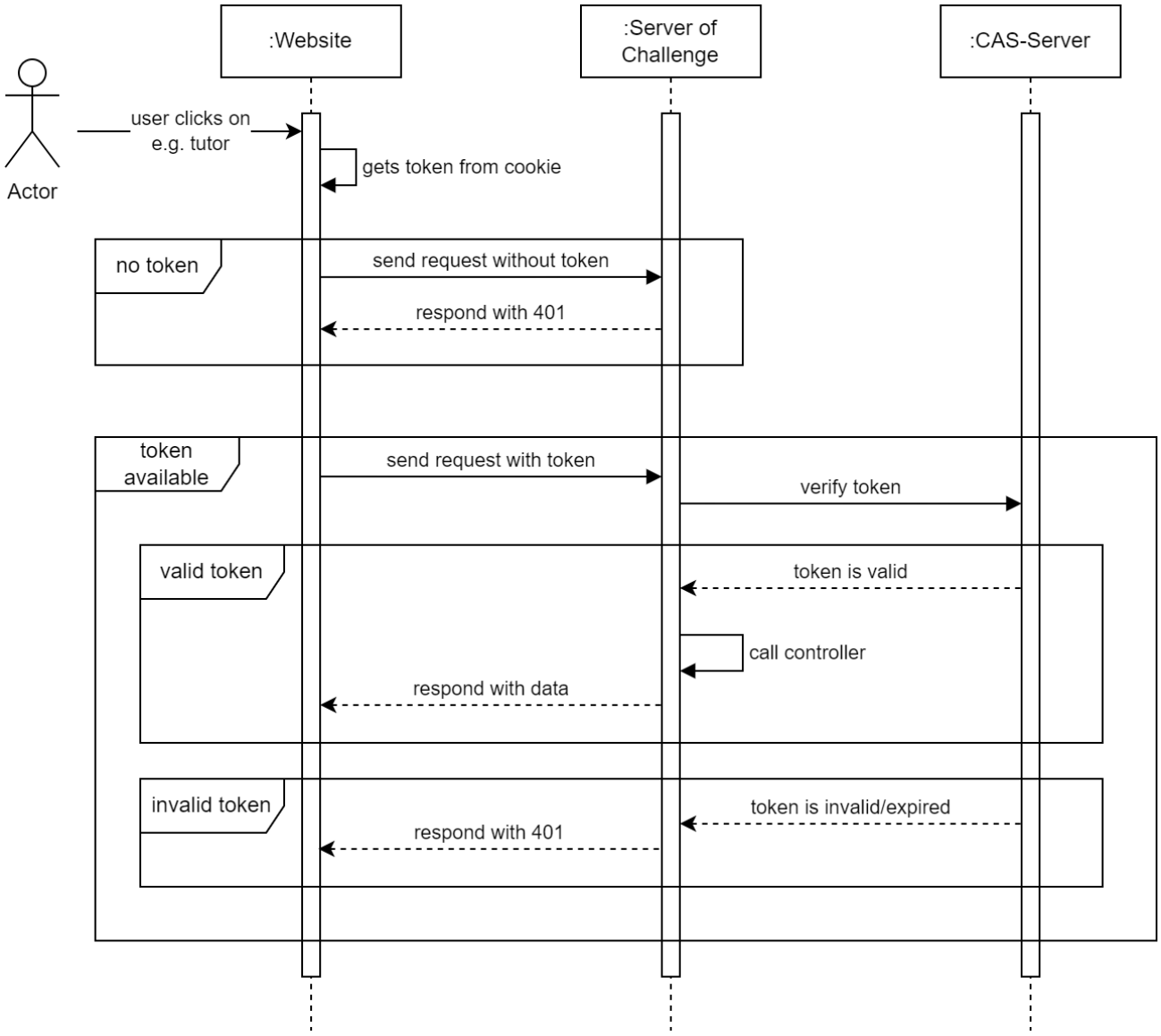


ABBILDUNG 6.5: Verifizierung des Tokens beim Spielen einer Challenge.

6.4 Challenges

Im folgenden Abschnitt wird die Architektur der einzelnen Challenges genauer erläutert. Dabei wird zuerst der allgemeine Aufbau beschreiben, da die Challenges alle gleich aufgebaut sind. In den danach folgenden Abschnitten wird auf konkrete Details eingegangen, die Challenge spezifisch sind.

6.4.1 Allgemeiner Aufbau

Dieser Abschnitt beschreibt den allgemeinen Aufbau einer Challenge. Dieser Aufbau dient als Grundlage für die restlichen Challenges. Zum allgemeinen Aufbau gehören folgende Punkte:

- **Allgemeiner Aufbau:** Beschreibt die Architektur der RestAPI und des Caches
- **Webseite:** Erläutert die Funktionsweise der Webseite
- **Docker-Container:** Visualisiert die Grundstruktur des Docker-Containers

Die Challenges wurden basierend auf dieser Architektur implementiert. Je nach Challenge mussten einige Module angepasst und/oder erweitert werden. Wie bereits in der Einleitung von Abschnitt 6.4 erwähnt, werden Challenges spezifischen Umsetzungen in dem jeweiligen Abschnitt erläutert.

6.4.1.1 RestAPI

Bei der Planung der Server-Architektur der RestAPI wurden verschiedene Aspekte beachtet. Da die Challenges alle auf dieser Architektur aufbauen, ist es besonders wichtig, dass die Erweiterbarkeit sowie die Trennung der Zuständigkeiten gewährleistet wird. Um eine effektive Trennung der Zuständigkeiten, auch bekannt als «separation of concerns», sicherzustellen, bietet sich eine Drei-Schichten-Architektur an. Ebenfalls ist diese Architektur für die Erweiterbarkeit gut geeignet, da aufgrund der drei Schichten von Natur aus eine lose Kopplung der Module verlangt wird. Bereits in früheren Projekten konnten positive Erfahrungen mit dieser Architektur gemacht werden und da sie sich auch für das aktuelle Projekt als geeignet erweist, wird sie hier ebenfalls eingesetzt.

Wie in Abbildung 6.6 zu sehen ist, bestehen diese drei Schichten (auch Layer genannt) aus einer Controller-, Service- und der Model-Schicht. Die Controller-Schicht ist dabei zuständig für die Abhandlung der eingehenden Requests. Die Service-Schicht wirkt als Abstraktionsebene zwischen der Model- und der Controller-Schicht. Die Model-Schicht selbst ist für die Datenanbindung verantwortlich.

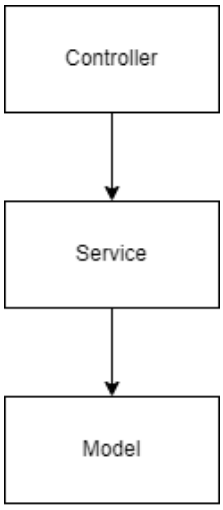


ABBILDUNG 6.6: Vereinfachte Darstellung der Drei-Schichten-Architektur.

Controller-Schicht

Wie in Abbildung 6.7 zu sehen ist, besteht die Controller-Schicht aus drei Komponenten. Zum einem aus dem Modul `Routes`, welches die eingehenden Anfragen and die richtigen Methoden der Controller-Schicht weiterleitet. Sie ist für das Routing des Servers zuständig. Zum anderen besteht diese Schicht aus den beiden Modulen «AuthController» und «LessonController». Der AuthController ist für die Sicherung der Challenges durch unbefugte Spieler*innen zuständig. Wie in Abbildung 6.5 visualisiert, überprüft der AuthController, ob die Anfrage weiter bearbeitet werden soll. Wenn die Anfrage vollständig und valide ist, dann wird sie an den LessonController weitergeleitet.

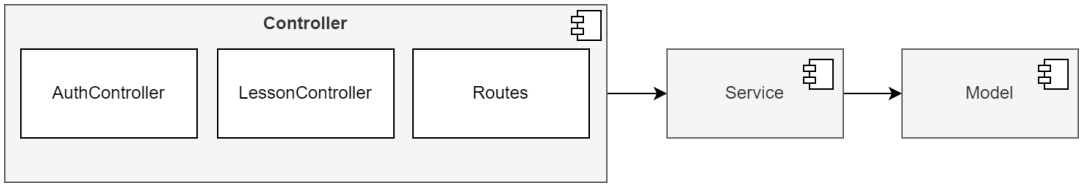


ABBILDUNG 6.7: Darstellung der Controller-Schicht des allgemeinen Aufbaus.

Der LessonController ist für die Abhandlung von Requests zuständig, welche Lektions-Daten anfragen. Zusätzlich dazu ist der LessonController verantwortlich für das Caching von einzelnen Requests. Um dies zu bewerkstelligen, kann der LessonController auf Funktionen des Service-Layers zurückgreifen. Diese Funktionen werden im folgenden Abschnitt genauer erläutert.

Service-Schicht

Die Service-Schicht besteht aus zwei Module die in Abbildung 6.8 zu erkenen sind.

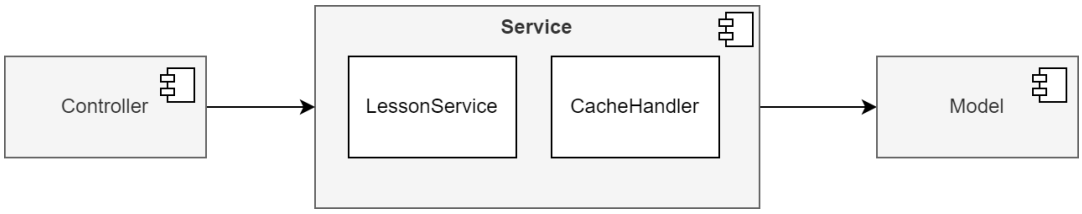


ABBILDUNG 6.8: Darstellung der Service-Schicht des allgemeinen Aufbaus.

Der LessonService greift auf die Model-Schicht zu und stellt der Controller-Schicht Funktionen zur Verfügung, mit welchen der Controller Daten aus der Model-Schicht abfragen kann. Der CacheHandler stellt Caching-Funktionalitäten zur Verfügung. Diese Funktionalitäten kann der Controller dann verwenden, um Requests zu Cachen oder Requests aus dem Cache zu holen.

Model-Schicht

Wie in Abbildung 6.9 zu sehen, besteht die Model-Schicht nur aus einem einzigen Modul, dem «DbMock».

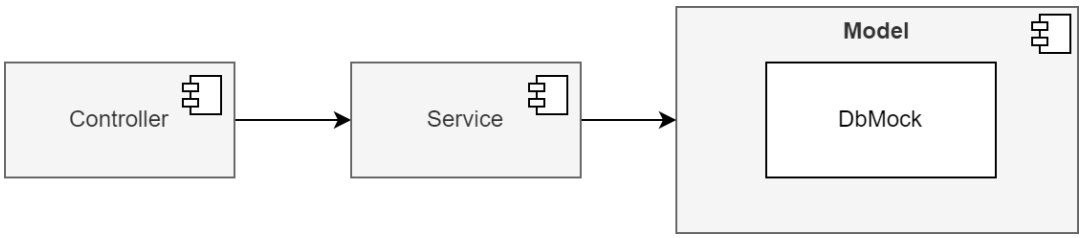


ABBILDUNG 6.9: Darstellung der Model-Schicht des allgemeinen Aufbaus.

Es wurde keine externe Datenbank verwendet. Die Daten, welche der Server bereit stellt, werden innerhalb des DbMock als Json-Object gespeichert. Dies wurde aus verschiedenen Gründen so implementiert. Der vorrangige Grund besteht darin, dass keine Ressourcen für die Erstellung und Wartung einer Datenbank aufgewendet werden müssen. Angesichts der geringen Datenmenge (zehn Datensätze) wäre es übertrieben, speziell für diese Zwecke eine eigene Datenbank zu erstellen, zu warten und zu hosten. Falls jedoch in Zukunft eine richtige Datenbank angebunden werden soll, stellt dies kein grosses Problem dar. In der Model-Schicht des Servers muss eine Anbindung zur Datenbank erstellt werden. Innerhalb dieser Anbindung müssen dann die in der Service-Schicht verwendeten Datenbank-Funktionen, welche mit den Daten aus der Datenbank interagieren, erstellt werden. Der Rest sollte weiterhin so funktionieren wie bisher.

Cache

Die RestAPI macht von einem internen Cache gebrauch. Dafür wird das «node-cache»-Modul [22] verwendet. Dieses ermöglicht es, den Cache einfach an die Challenge spezifischen Anforderungen anzupassen. Im Abschnitt zur Service-Schicht wird der CacheHandler erwähnt. Er kümmert sich um das Caching. In seiner Default Konfiguration werden nur die URL und der HOST als Cache-Key verwendet.

API Documentation

Der allgemeine Aufbau stellt über die RestAPI zwei Routen zur Verfügung (`lessons` und `singleLesson`). Für detailliertere Informationen kann die API-Dokumentation im Anhang (siehe A.1) konsultiert werden.

6.4.1.2 Webseite

Die Planung des Webseiten-Themas und seines Erscheinungsbildes wurde bereits in Kapitel 5.2 erläutert. Die einfachste Version der Webseite besteht aus zwei Seiten, einer Home-Seite (siehe Abbildung 5.1) und einer Detail-Seite (siehe Abbildung 5.2). In diesem Abschnitt wird auf den Aufbau der Webseite genauer eingegangen.

Funktionsweise

Um Sicherheitslücken einfach einzubinden, wird vieles in JavaScript gelöst. Daher wurde ein kleines Framework entwickelt, das eine einfache Kommunikation mit der RestAPI ermöglicht. Ziel dieses Frameworks ist es mit möglichst wenig Aufwand Erweiterungen, die bei der RestAPI entstanden sind, in die Webseite zu übernehmen, ohne dass der bisheriger Code geändert werden muss.

Abbildung 6.10 visualisiert die Funktionsweise des entwickelten Frameworks. Das zentrale Element bildet die `http.js` Komponente. Sie stellt eine `GET()` und eine `POST()` Funktion zur Verfügung, die Requests an die RestAPI schicken.

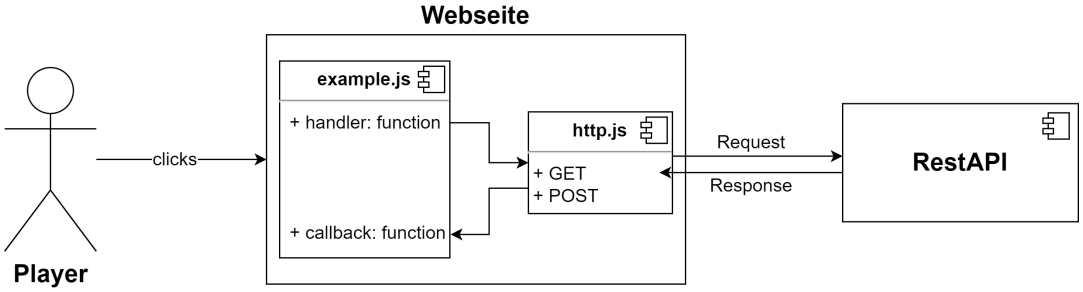


ABBILDUNG 6.10: Funktionsweise des entwickelten Frameworks für zur Kommunikation mit der RestAPI.

Die Details, welche für eine Request benötigt werden, werden mittels eines JSON-Objekts der Funktion übergeben. Code 6.1 zeigt dies anhand der Implementierung der GET-Funktion. Dieses Objekt kann zwei optionale Callbacks beinhalten, welche im Falle eines Erfolgs resp. Fehlers aufgerufen werden. Die aufrufende Funktion besitzt somit die vollständige Kontrolle über den Programmfluss.

```
/**
 * Send a GET request to the desired URL.
 *
 * @param {object}          properties - Object which contains all
   required information.
 * @param {string}          properties.URL - The URL for the request.
 * @param {function(data)}  properties.success - Callback which is
   called, if the request was successful (Status Code 200-299)
 * @param {function(data)}  properties.error - Callback which is called
   , if the request was not successful
 */
export const GET = (properties) => {
    fetch(properties.URL, {
        method: 'GET',
        .
        .
        .
    }).then((response) => {
        return response.ok
            ? response.json()
            : properties?.error?.(response)
    }).then((data) => properties?.success?.(data))
        .catch((data) => properties?.error?.(data))
}
```

CODE 6.1: Funktionsbeschreibung der GET-Funktion.

In Abbildung 6.10 klickt ein*e Spieler*in auf ein Element der Webseite. Dies kann bspw. das Bild eines*einer Tutors*Tutorin sein. Die damit verbundene Funktion (hier `handler` genannt) würde eine Anfrage an die RestAPI schicken. Dabei übergibt sie eine Callback-Funktion `callback` die aufgerufen werden soll, wenn die Antwort positiv ausfällt.

Aufbau

Die beiden Seiten Home und Detail sind dynamisch aufgebaut. Der Inhalt wird über die RestAPI abgefragt und anschliessend angezeigt. Dies geschieht mithilfe des im vorherigen Abschnitt beschriebenen Frameworks. Was alle Seiten gemeinsam haben ist, dass sie eine Navigationsleiste besitzen. Diese wird allen Webseiten by default hinzugefügt. Abbildung 6.11 zeigt, wie eine Seite aufgebaut ist inkl. des Navigationsheaders. Die grüne Box symbolisiert dabei den Inhalt, welcher für jede Seite individuell erstellt wird.

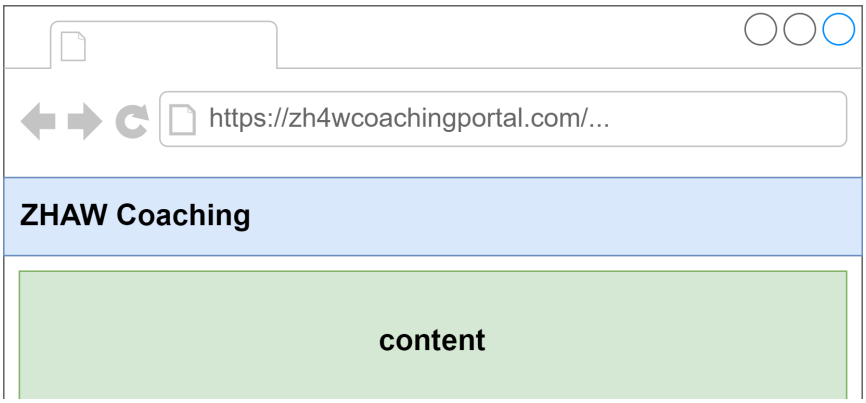


ABBILDUNG 6.11: Allgemeine Struktur einer Seite.

6.4.1.3 Docker-Container

Wie bereits im Abschnitt zur verwendeten Technologie (siehe 6.1) erwähnt, wird Nginx als Host für die Webseite und als Reverse-Proxy für die RestAPI verwendet. Für die Containerisierung wird auf Docker gesetzt. Dadurch ist es möglich, ein Image für jede Challenge zu erstellen, welches dann dem C2-Server erlaubt, beliebig viele Container einer Challenge zu starten.

Abbildung 6.12 zeigt, wie die interne Struktur eines Containers aussieht, der nur RestAPI und Webseite besitzt.

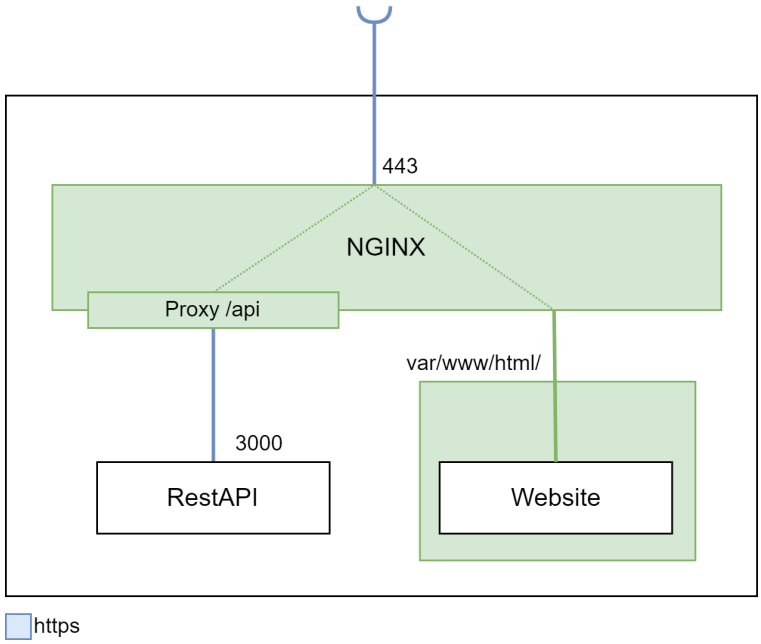


ABBILDUNG 6.12: Struktur des Containers der Challenge How To Web Cache Poison.

Natürlich wäre es auch möglich gewesen, die RestAPI über einen separaten Port zur Verfügung zu stellen. Dies würde jedoch bedeuten, dass jede Challenge mindestens zwei Ports beanspruchen würde. Um Platz zu sparen, wurde daher entschieden, dass die RestAPI und Webseite über einen gemeinsamen Port erreichbar sind und alle Anfragen, die mit `/api` beginnen, werden auf die RestAPI umgeleitet.

6.4.2 Easy: How To Web Cache Poison

Die Challenge Easy orientiert sich am allgemeinen Aufbau. Zusätzlich musste jedoch noch die Sicherheitslücke in der RestAPI sowie der Webseite hinzugefügt werden. Ebenfalls musste die Schritt-für-Schritt Anleitung auf der Webseite zugänglich gemacht werden. Die Anleitung ist in der Navigationsleiste vorzufinden.

In diesem Abschnitt wird auf folgende Punkte genauer eingegangen:

- Aufbau eines Docker-Containers
- Eingefügte Sicherheitslücke in der RestAPI
- Eingefügte Sicherheitslücke in der Webseite

6.4.2.1 Docker-Container

Abbildung 6.12 zeigt den Aufbau eines Docker-Containers. Da die Challenge Easy nur eine Webseite mit dazugehöriger RestAPI besitzt, musst der Docker-Container nicht angepasst werden.

6.4.2.2 Sicherheitslücke in der RestAPI

Damit ein Web Cache Poisoning Angriff möglich wird, muss die RestAPI einen unkeyed Header verwenden. Weiter muss dieser Header einen Einfluss auf die Response haben, die von der Webseite verarbeitet wird. Für den Angriff musste der Cache nicht angepasst werden, nur die Controllerfunktion wurde geändert.

Code 6.2 visualisiert wie der unkeyed Header verwendet wird. Wenn das Bild auf der Detail-Seite geladen wird, wird der Header `x-forwarded-host` (orange markiert) dazu verwendet, um den Pfad für das Bild, welches angezeigt werden soll, zu definieren.

```
let lesson = this.cacheHandler.getByRequestHeaders(req)

if (lesson) {
    res.setHeader('X-Cache', 'HIT');
    return res.status(200).json(lesson);
}

lesson = await this.service.getById(req.params.id)
if (lesson) {
    lesson.image = req.headers['x-forwarded-host'] + lesson.image;
    this.cacheHandler.cacheObject(req, lesson)
    res.setHeader('X-Cache', 'MISS');
    return res.status(200).json(lesson)
}
```

CODE 6.2: Sicherheitslücke im Programmfluss des LessonControllers.

Dies Umsetzung führt dazu, dass Werte aus dem unkeyed Header in der Response reflektiert werden können. Das bedeuted, wenn beispielsweise der Wert von `lesson.image /image/path.png` enthält und der `x-forwarded-host` Header den Wert

`https://evilsite.com`

besitzt, der Bildpfad in der Response anschliessend wie folgt aussehen würde:

`https://evilsite.com/image/path.png`

Falls der `x-forwarded-host` nicht gesetzt wird, bleiben die Bildfpade trotzdem gültig. Die Bilder werden in diesem Fall immernoch auf der Webseite angezeigt. Der Server ist nun mittels Web Cache Poisoning angreifbar.

6.4.2.3 Sicherheitslücke der Webseite

Damit der Wert aus der Response auf eine gefährliche Weise verwendet werden kann, muss auch die Website vorbereitet werden. Andernfalls kann die Schwachstelle des Servers nicht ausgenutzt werden. Code 6.3 zeigt, wie dies umgesetzt wurde.

```
document.getElementById("image-placeholder").innerHTML +=
`<img src="${lesson['image']}">`;
```

CODE 6.3: Sicherheitslücke in der createSingleLesson-Funktion des singleLesson.js Files.

Dieser Code ermöglicht es, dass die Quelle des Bildes auf eine gefährliche Art und Weise gesetzt wird. Denn so ist es möglich, aus dem img-tag auszubrechen und JavaScript-Code auszuführen. Wenn beispielsweise folgendes im image-Attribut stehen würde: `#" onerror=alert(1)>/image/path.png`, würde ein alert auf der Webseite angezeigt werden.

6.4.3 Medium: Cross Site Cookie Mining

Die Medium-Challenge baut auf der Easy-Challenge auf. Zusätzlich zur Webseite und dem Server kommt ein Request-Catcher hinzu. Die Änderungen in der Architektur beziehungsweise dem Aufbau sowie die Anpassungen an den Schwachstellen werden in diesem Abschnitt erläutert. Des Weiteren wird die Funktionsweise des Docker-Containers aufgezeigt.

6.4.3.1 Docker-Container

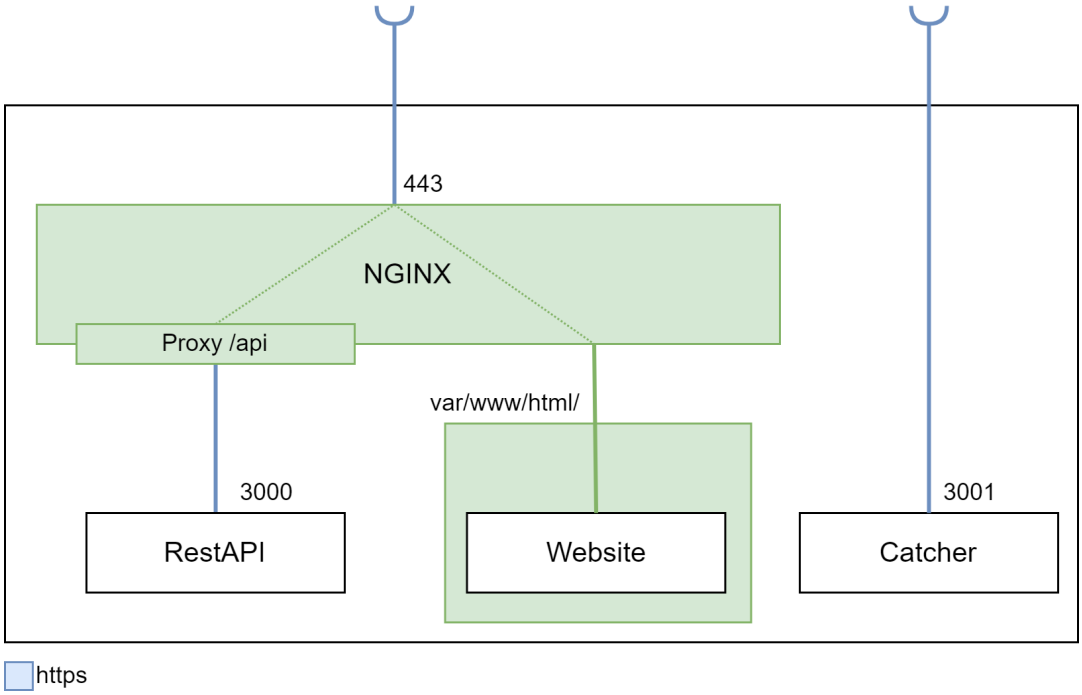


ABBILDUNG 6.13: Struktur des Containers der Challenge Cross Site Cookie Mining.

Abbildung 6.13 zeigt den Aufbau eines Docker-Containers. Jedes Image der Challenge stellt zwei Ports zur Verfügung, über diese die verschiedenen Services erreichbar sind. Der Aufbau der RestAPI und der Webseite mit Nginx unterscheidet sich nicht vom allgemeinen Aufbau (siehe 6.4.1.3). Neu hinzugekommen ist der Catcher.

6.4.3.2 RestAPI

In diesem Abschnitt werden die Änderungen am Aufbau und der Sicherheitslücken der RestAPI erläutert. Die vorgenommenen Änderungen betreffen ausschliesslich die Controller-Schicht. Es wurden keine weiteren Modifikationen in anderen Teilen der RestAPI vorgenommen.

Controller-Schicht

Wie in Abbildung 6.14 zu sehen ist, wurde der Controller-Schicht ein LoginController hinzugefügt. Dieser Controller bietet Funktionen für das Login an. Einerseits sind dies Funktionen, um sich ein- beziehungsweise auszuloggen und andererseits muss er sich auch um die Validierung der Session und der Cookies kümmern.

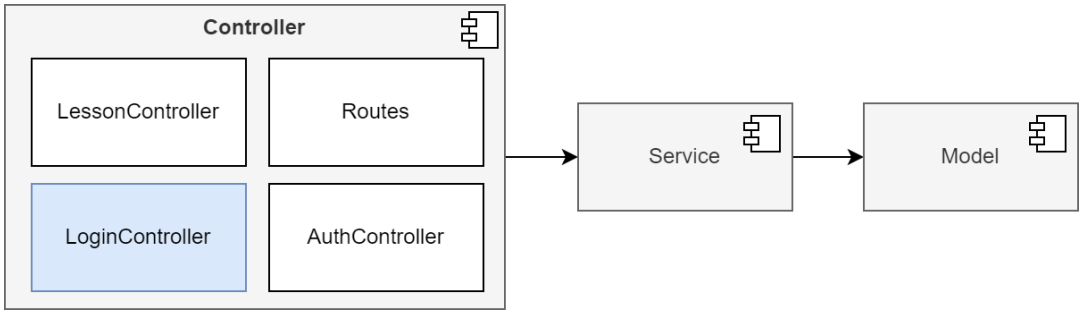


ABBILDUNG 6.14: Darstellung der Controller-Schicht der Medium Challenge.

Diese Sessions werden mithilfe des von Express zur Verfügung gestellten Moduls `express-session` erstellt. Eine Session ist die Zeit zwischen der Anmeldung und der Abmeldung einer spielenden Person. In dieser Zeit, sofern die Session noch gültig ist, muss sie sich nicht erneut anmelden. Das Token, welches für die Session erstellt wird, wird in einem Cookie gespeichert. Für das Setzen dieses Cookies ist ebenfalls der LoginController verantwortlich.

Eine weitere Änderung in der Controller-Schicht befindet sich im Modul Routes. Dort wurden die Routen zu den angebotenen Funktionen des LoginControllers hinzugefügt.

Die letzte Änderung wurde im LessonController durchgeführt. Dort wird nun, mithilfe der Funktionen des LoginController, überprüft, ob eine Session gültig ist oder nicht. Wenn nicht, kann die spielende Person nicht auf die Detail Seiten zugreifen.

Service- und Model-Schicht

Da lediglich der LoginController dazu kam und dieser keine Daten aus der Model-Schicht benötigt, wurden auch keine Änderungen an der Model- beziehungsweise der Service-Schicht vorgenommen. Diese beiden Layer stimmen mit der Architektur des im Abschnitt 6.4.1.1 erläuterten Servers überein.

Sicherheitslücken

Das Ziel der Challenge ist es, ein Session-Cookie eines Bots zu stehlen und so an das Token zu kommen (siehe Abschnitt 5.1.2). Damit dies ermöglicht wird, mussten Anpassungen an den Sicherheitslücken vorgenommen werden.

Die Funktionsweise der Sicherheitslücke unterscheidet sich nicht von der Easy Challenge, jedoch wurde der unkeyed Header angepasst. In der Medium Challenge wird der `x-images` Header anstelle des `x-forwarded-host` Headers verwendet, wie im Code-Ausschnitt 6.4 (orange markiert) zu sehen ist. Ausserdem ist neu in der Response zu sehen, wie lange die Response im Cache bleibt (grün markiert). Da ein Bot die Webseite regelmässig durchsucht, landen auch seine Anfragen im Cache. Durch

diese Änderung sieht der*die Spieler*in, wie lange er*sie warten muss, falls der Bot schneller war und dessen Anfrage im Cache gelandet ist.

```
if (lesson) {
    let header = req.headers['x-images'];
    lesson.image = header ? header + lesson.image : lesson.image;
    lesson.CachedOn = Date.now()
    this.cacheHandler.cacheObject(req, lesson)
    res.setHeader('max-age', this.config.server.cache.ttl)
    res.setHeader('X-Cache', 'MISS');
    return res.status(200).json(lesson)
}
```

CODE 6.4: Sicherheitslücke in der getById-Funktion des LessonControllers.

Da dies kein üblicher Header ist, müssen die Spielenden viel ausprobieren, bis sie einen unkeyed Header finden oder auf ein Brute-Force-Tool, wie beispielsweise der Param-Miner von Burp, zurückgreifen. Diese Art von Sicherheitslücke kann vorkommen, wenn sich Anforderungen an der Software ändern, jedoch nicht richtig umgesetzt wurden.

Weiter musste das Session Cookie so konfiguriert werden, dass es sich mittels JavaScript-Injection stehlen lässt. Um dies zu ermöglichen, wurde es folgendermassen konfiguriert:

```
app.use(sessions({
    secret: process.env.SESSIONSECRET,
    saveUninitialized: true,
    cookie: { maxAge: oneDay, sameSite: 'none', secure: true, httpOnly: false },
    resave: false
}))
```

CODE 6.5: Sicherheitslücke in der Konfiguration des Session-Cookies.

Wie in Code 6.5 in orange markiert zu erkennen ist, wurde `httpOnly` auf `false` und `sameSite` auf `'none'` gesetzt. Wenn das `httpOnly`-Flag auf `false` gesetzt wird, erlaubt es dem Browser, dass mittels JavaScript auf das Cookie zugegriffen werden kann. Da das `sameSite`-Flag auf `'none'` gesetzt wurde, ermöglicht dies eine Weiterleitung des Cookies an andere Webseiten. Diese beiden Schwachstellen erlauben den Spielenden nun, das Cookie mittels JavaScript auszulesen und es an eine andere Seite zu schicken.

Änderungen an API

Die RestAPI wurde um zwei zusätzliche Post-Routen (`login` und `logout`) erweitert. Für detailliertere Informationen zu diesen Routen kann die API-Dokumentation im Anhang (siehe A.1) konsultiert werden.

6.4.3.3 Webseite

Durch die Einführung eines Session-Cookies wurde auch die Webseite erweitert. Dafür mussten zwei Änderungen vorgenommen werden:

1. Es wurde eine Login-Seite erstellt. Diese besteht aus einem einfachen Formular, dass Benutzername und Passwort an die RestAPI sendet.
2. Klickt ein*e Spieler*in auf einen*eine Tutor*in und ist nicht angemeldet, wird der*die Tutor*in nicht angezeigt.

Die Login-Funktionalität wurde in die Navigationsleiste eingebaut.

Bezüglich der Sicherheitslücke musste nichts unternommen werden. Da sie auch auf XSS basiert und ebenfalls beim Laden der Bilder eingeschleust werden kann, konnte das gleiche Modul wie bei der Challenge Easy verwendet werden (siehe Abschnitt 6.4.2.3).

6.4.3.4 Catcher

Wie in Abschnitt 5.1.2 erwähnt, dient der Catcher als Auffangnetz von Requests. D. h. alle Requests, die an den Catcher gesendet werden, werden auf einer vorgefertigten Seite dargestellt. Er dient als Hilfsmittel zum Lösen der Challenge.

Der Catcher besteht aus einem Server, der statische HTML Dateien liefert. Requests die an den Catcher geschickt werden, werden über eine unidirektionale Verbindung vom Server zur Webseite angezeigt. Abbildung 6.15 visualisiert die Kommunikation des Servers mit der Webseite.

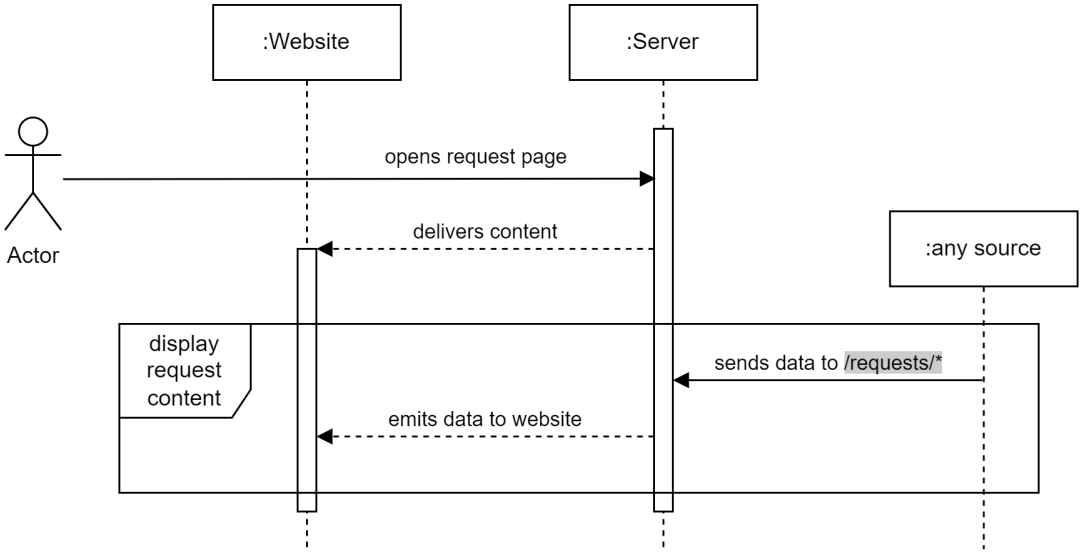


ABBILDUNG 6.15: Kommunikation zwischen Server und Webseite des Catchers.

Die nachfolgenden beiden Abschnitte beschreiben, wie dies für den Server und die Webseite umgesetzt wurde.

Server

Für das Routing sowie das Speichern und weiterleiten der empfangen Requests ist der Server zuständig.

Der Server besitzt drei Routen.

- `GET "/"`: Liefert die Startseite des Catchers. Sie beschreibt, wie der Catcher funktioniert.
- `GET "/requests"`: Liefert die Requestseite, auf welcher alle abgefangenen Requests angezeigt werden.

- `ALL "/requests/*"`: Requests, die an diese Route geschickt werden, werden gespeichert und anschliessend auf der Requestseite angezeigt. `ALL` steht dafür, dass jegliche Requests (`GET`, `POST`, `PUT` usw.) akzeptiert werden.

In Abbildung 6.15 ist der Prozess `display request content` enthalten. Wenn jemand die Requestseite öffnet, ist diese zu Beginn leer. Alle Requests, die anschliessend an `/requests/*` gesendet werden, werden mittels eines Socket-Events an die Requestseite übermittelt.

Webseite

Wie oben erwähnt, wird über die beiden `GET`-Routen jeweils eine statische HTML Seite geliefert. Die Startseite enthält eine kurze Anleitung für den Catcher und zusätzlich eine Verlinkung zur Requestseite. Die Requestseite stellt die abgefangenen Requests dar. Dies hat den Vorteil, dass die Requestseite nicht ständig manuell von dem*der Benutzer*in neu geladen werden muss, um sie auf neue Requests zu überprüfen. Stattdessen wird die Aktualisierung durch Socket-Events automatisch durchgeführt.

Sobald die Requestseite geladen wird, stellt sie eine Verbindung mit einem Socket her. Code 6.6 zeigt, wie dies umgesetzt werden kann. Dabei wird das `request`-Event abonniert, welches vom Server für die Kommunikation verwendet wird. Wenn das Event ausgelöst wird, erhält die Requestseite alle abgefangenen Requestobjekte, welche dann dargestellt werden. Durch diese Vorgehensweise wird sichergestellt, dass alle auf dem Server empfangenen Requests schnell und effizient dargestellt werden können, ohne dabei umständliche oder komplizierte Prozesse durchlaufen zu müssen.

```
1 let socket = io();
2 socket.on('request', function (requests) {
3     // update request data on page
4 });
```

CODE 6.6: Implementierung des Socket-Listeners auf der Webseite.

6.4.3.5 Login-Bot

Da Web Cache Poisoning darauf basiert, dass die vergifteten Seiten von anderen Benutzenden aufgerufen werden, wurde ein Bot entwickelt, der diese Verhalten simuliert. Im Bereich des Softwaretestings von Webanwendungen wird gerne von Selenium gebrauch gemacht. Mit Selenium können Interaktionen auf einer Webseite automatisiert werden. Damit lässt sich ideal ein Bot entwickeln, der einen*eine Benutzer*in simulieren soll.

Aufgabe

Die Aufgaben des Login-Bot sind folgende:

1. Meldet sich als Administrator auf der Webseite an.
2. Öffnet alle Detail-Seiten in regelmässigen Abständen, um damit die potenziell vergiftete Seite zu öffnen.

Umsetzung

Wie bereits erwähnt, wird Selenium für die Umsetzung gebraucht. Das Script zur Durchführung des Prozesses wurde in Python erstellt. Damit der Login-Bot in einem Container ausgeführt werden kann, müssen jedoch ein paar Konfigurationen vorgenommen werden. Code 6.7 listet alle vorgenommenen Konfiguration auf, die benötigt werden, damit der Login-Bot in einem Container ausgeführt wird.

```
options = Options()
options.add_argument("--headless=new")
options.add_argument("--window-size=1920,1080")
options.add_argument("--no-sandbox")
options.add_argument('--disable-dev-shm-usage')
options.add_argument("--ignore-ssl-errors=yes")
options.add_argument("--ignore-certificate-errors")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install
    ()), options=options)
```

CODE 6.7: Konfiguration des Login-Bots zur Ausführung in einem Container.

Die Konfigurationen werden aus folgenden Gründen benötigt:

`"--headless=new"`: Sorgt dafür, dass kein Browserfenster geöffnet wird, da das Script in einem Container ausgeführt wird und dadurch kein GUI zur Verfügung hat.

`"--window-size=1920,1080"`: Obwohl kein Fenster sichtbar ist, müssen alle Elemente für Selenium virtuell sichtbar sein, dafür wird dem Browserfenster eine fixe Grösse gegeben.

`"--no-sandbox"`: Wenn Chrome als Root-Benutzer gestartet wird, besteht die Chance unter Linux, dass der WebDriver crashed. Diese Option verhindert diesen Fehler.

`"--disable-dev-shm-usage"`: Der Container besitzt eine limitierte Anzahl Ressourcen. Diese Option verhindert, dass Chrome die Dateien im Shared Memory ablegt.

Die beiden `"--ignore-*` Optionen werden gebraucht, da momentan noch ein Self-Signed Certificate verwendet wird.

6.4.4 Hard: Double Cache Mayhem

Die Hard-Challenge baut auf der Medium-Challenge auf. Zusätzlich zur Webseite, der RestAPI und dem Catcher wurde ein externer Cache sowie eine MITM-Seite hinzugefügt. Die Änderungen in der Architektur beziehungsweise dem Aufbau sowie die Anpassungen an den Schwachstellen werden in diesem Abschnitt erläutert.

6.4.4.1 Docker-Container

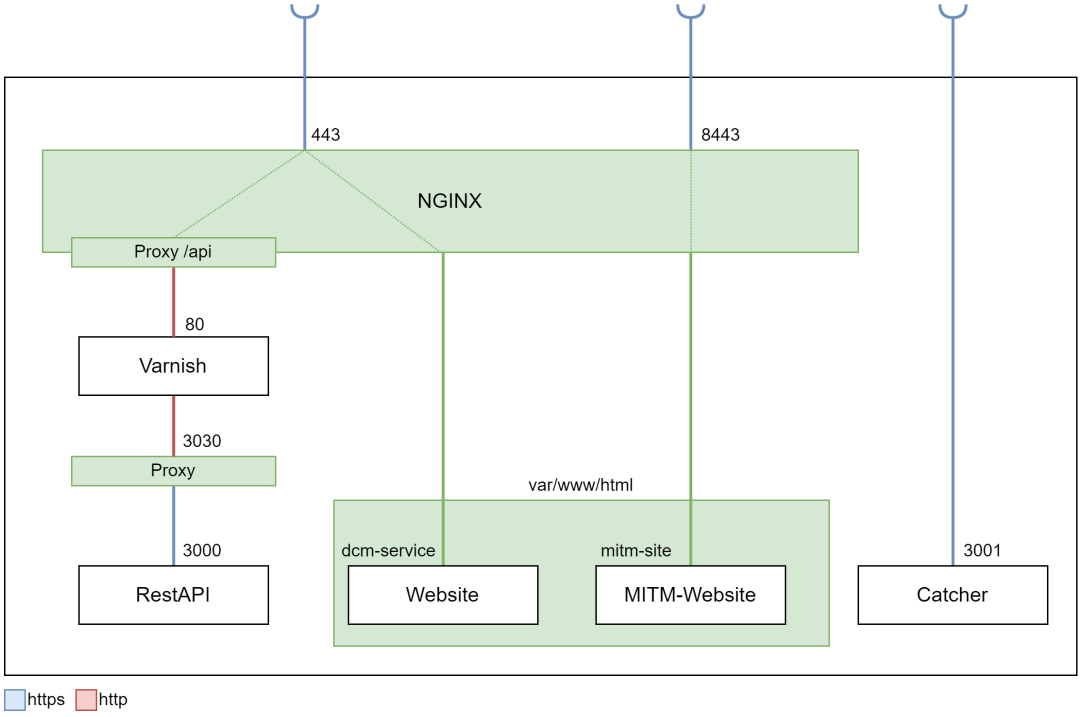


ABBILDUNG 6.16: Struktur des Containers der Challenge Double Cache Mayhem.

Abbildung 6.16 zeigt den Aufbau eines Docker-Containers. Jedes Image der Challenge stellt drei Ports zur Verfügung, über diese die verschiedenen Services erreichbar sind. Der Aufbau des Containers basiert auf dem Container der Challenge Medium. Neu hinzugekommen ist die MITM-Webseite, die über einen eigenen Port erreichbar ist. Sie wird ebenfalls von Nginx gehosted. Ebenfalls neu ist der Varnish-Cache. Da Varnish kein TLS unterstützt, wird Nginx als Proxy vor und nach Varnish eingesetzt. Dabei wird die Kommunikation von HTTPS nach HTTP umgeleitet und anschliessend wieder von HTTP nach HTTPS. Das gleich wiederholt sich bei ausgehenden Responses von der RestAPI.

6.4.4.2 RestAPI

Die RestAPI der Hard-Challenge basiert auf der RestAPI der Medium-Challenge. Wie schon bei der Medium-Challenge mussten auch hier Änderungen in der Contoller-Schicht vorgenommen werden. Die eingebaute Sicherheitslücke der Medium Challenge musste entfernt werden und die neue Sicherheitslücke wurde eingebaut.

Änderungen an der Sicherheitslücken

In Abbildung 6.17 (orange markiert), sind die von den Änderungen betroffenen Module der Controller-Schicht zu sehen.

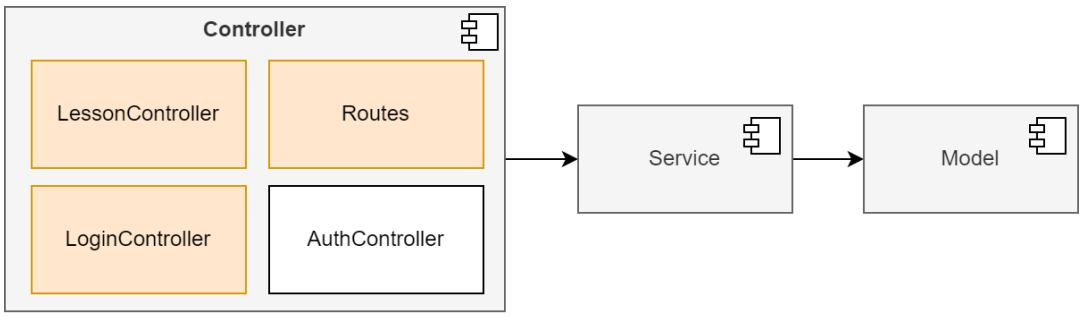


ABBILDUNG 6.17: Darstellung der Controller-Schicht der Medium Challenge.

Da die Sicherheitslücke keine JavaScript-Injection beinhaltet, besteht keine Notwendigkeit mehr, dass der LessonController Anfragen zwischenspeichert. Aus diesem Grund wurde das Caching innerhalb des Controllers entfernt. Dadurch ist dieser Controller auch nicht mehr anfällig für Web Cache Poisoning Angriffe.

Wie im Abschnitt 5.1.3 beschrieben, ist das Ziel der Challenge eine MITM-Attacke durchzuführen. Zu diesem Zweck wurde dem Login Controller die Funktion `getLogin` hinzugefügt. Diese Methode gibt in der Response die Location der `login.html`-Seite zurück. Anfragen an diese URL wurden dem CacheHandler hinzugefügt. Dadurch können die spielenden Personen den Cache bzw. diese Location manipulieren. Wenn diese manipulierte Request dann an andere spielende Personen gesendet wird (aus dem Cache), erhalten auch sie die falsche Location und könnten beispielsweise auf die MITM-Seite weitergeleitet werden.

In der Service Schicht musste nur der Cache Handler angepasst werden. Die Sicherheitslücke CVE-2018-14773 [8] dokumentiert, das es möglich war den Cache-Key zu überschreiben. Diese Verhaltensweise wurde ebenfalls in die RestAPI eingebaut.

Bei der Generierung des Cache-Keys wird nun geprüft, ob dieser Header vorhanden ist. Ist dies der Fall, wird der gesamte bisher erstellte Cache-Key mit dem im Header enthaltenen Wert überschrieben. Dies konnte daher so implementiert werden, da nur in der `getLogin`-Funktion gecached wird. Würde an anderen Stellen, wie zum Beispiel im LessonController, gecached werden, müsste der Host-Header im Cache-Key belassen werden.

Code 6.8 zeigt, dass der CacheHandler den Cache-Key überschreibt, wenn der Header `x-original-url` mitgesendet wird.

```
let currentHeaderValue = reqHeaders[key];
if(key == "x-original-url" && currentHeaderValue){
    cacheKey = currentHeaderValue
    return
}
```

CODE 6.8: Sicherheitslücke in der generateCacheKey-Funktion des CacheHandler.

Änderungen an API

Die RestAPI wurde um die `getLogin`-Route erweitert. Für detailliertere Informationen kann die API-Dokumentation im Anhang (siehe A.1) konsultiert werden.

6.4.4.3 Webseite

In diesem Abschnitt werden die Änderungen am Webseitenaufbau und den Sicherheitslücken erläutert. wie bereits im Abschitt zur RestAPI erwähnt, basiert die Challenge Hard auf der Medium Challenge. Daher wurde auch Webseite von der Medium Challenge übernommen und entsprechend Angepasst.

Anpassung der Sicherheitslücke

Die JavaScript-Injection-Schwachstelle, welche in den beiden vorherigen Challenges verwendet wurde, wird nicht mehr benötigt, weshalb sie aus der Webseite entfernt wurde.

Wie in Abschnitt 5.1.3 beschrieben, soll es möglich sein, eine MITM-Seite einzuschleusen. Wenn beim Klicken auf den Login-Button direkt zur Login-Seite verwiesen wird, besteht keine Möglichkeit, dies durchzuführen. Die Webseite sendet neu eine Request an `getLogin`, um die Location der Login-Seite zu erhalten. Diese Location, die aus der Response gelesen wird, wird dann für die Weiterleitung zur Login-Seite verwendet.

Wenn die spielenden Personen erfolgreich dafür sorgen können, dass diese Location angepasst wird, kann damit auf die MITM-Seite verwiesen werden, was letztendlich zur Lösung der Challenge führen würde.

6.4.4.4 Varnish

Als externer Cache wird Varnish verwendet. Die Entscheidung für Varnish basiert darauf, dass die Challenge Hard auf einer Sicherheitslücke beruht, die in Drupal existierte und Varnish ein weitverbreiteter HTTP-Beschleuniger für Drupal-Seiten ist [10] [23].

Wie bereits im Konzept erwähnt dient der Cache lediglich einer weiteren Hürde, die überwunden werden muss (siehe Abschnitt 5.1.3). Jedoch musste der Cache in siner Standard-Konfiguration angepasst werden, da Varnish bestimmte Requests standardmäßig nicht cached aus folgenden Gründen:

- Sieht Varnish eine Request, die ein Cookie beinhaltet, wird diese nicht gecached [24].
- Wenn Varnish einen `Authorization` Header sieht, wird die Request nicht im Cache aufgenommen [25].

Damit die Sicherheitslücke funktioniert muss die GET-Request an `/getLogin` im Cache aufgenommen werden. Ausserdem soll die Request gecached werden, obwohl ein `Authorization` Header vorhanden ist.

Ausserdem wurden der Response ein `X-Cache: HIT/MISS` hinzugefügt, damit Erkennbar ist, ob die Response aus dem Varnish Cache stammt oder nicht.

6.4.4.5 MITM-Seite

Die Man-in-the-Middle-Seite dient den Spielenden als Hilfestellung, indem sie ihnen eine bereits vorhandene MITM-Webseite zur Verfügung stellt. Bei einem MITM-Angriff ist es normalerweise erforderlich, eine Webseite zu replizieren und potenzielle Opfer auf diese gefälschte Seite zu leiten. Diesen Schritt müssen die Spieler*innen nicht vornehmen und das aus zwei Gründen:

1. Es vereinfacht die Lösung der Herausforderung für die Spielenden, da sie keine eigene Webseite erstellen müssen. Die MITM-Seite ist ausserdem ein erster kleiner Hinweis.
2. Es ermöglicht den Einsatz des Login-Bots. Wenn den Spielenden keine vorgefertigte Seite zur Verfügung gestellt würde und sie die Seite selbst schreiben müssten, könnte der Login-Bot nicht wissen, wie die Buttons benannt wurden und sich folglich nicht einloggen. Dies würde bedeuten, dass die Herausforderung nur lösbar wäre, wenn die Spielenden selbst darauf kämen, dass ihre Seite genau die gleichen Namen für die Elemente benötigt - was unrealistisch wäre.

Home-Seite

Die Home-Seite der MITM-Seite ist eine exakte Kopie der Login-Seite der Medium Challenge. Einziger Unterschied besteht darin, dass die CSS-Datei angepasst wurde, damit die Seite von der normalen Login-Seite visuell zu unterscheiden ist.

Wie in Abbildung 6.18 zu sehen, verwendet die Seite zwei Javascript Files.

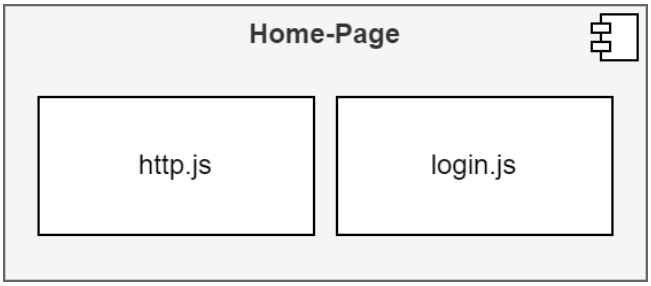


ABBILDUNG 6.18: Verwendete JavaScript Files der MITM Home-Seite.

Zum einen verwendet die Seite die `http.js`-Datei, die unverändert aus den vorherigen Challenges übernommen wurde, und zum anderen die `login.js`-Datei, die im Vergleich zur Medium Challenge angepasst wurde. Die `login.js`-Datei wurde modifiziert, so dass sie die eingegebenen Benutzerdaten nicht mehr an den Server der Challenge sendet, sondern an den Request-Catcher.

Nachdem die Benutzerdaten übertragen wurden, wird der*die Spieler*in auf die Hacked-Seite weitergeleitet.

Wenn also die MITM-Seite anstelle der normalen Login-Seite geladen wird und sich der Login-Bot anmeldet, werden seine Benutzerdaten an den Catcher gesendet und die Challenge ist gelöst.

Hacked-Seite

Die Hacked-Seite hat reinen Unterhaltungswert. Wenn die MITM-Seite erfolgreich eingeschleust wurde und sich eine spielende Person anmelden möchte, wird die MITM-Home-Seite angezeigt. Sobald die Benutzerdaten eingegeben werden, erfolgt eine Weiterleitung zur Hacked-Seite. Auf dieser Seite wird eine Nachricht angezeigt, die besagt, dass man «gehackt» wurde, begleitet von einem kurzen GIF. Zusätzlich wurde ein Link auf der Hacked-Seite eingefügt, um zur normalen Challenge-Seite zurückzukehren.

6.4.4.6 Catcher

Am Catcher wurden keinerlei Änderungen vorgenommen. Die Architektur stimmt mit derjenigen überein, die in Abschnitt 6.4.3.4 beschriebenen wurde.

6.4.4.7 Login-Bot

Auch bei der Hard Challenge wird ein Login-Bot verwendet, der auf dem Bot der Medium Challenge basiert. Da es in der Hard Challenge nicht erforderlich ist, die einzelnen Detail-Seiten zu besuchen, wurde diese Funktionalität aus dem LoginBot entfernt. Ansonsten ist der Login-Bot prinzipiell unverändert geblieben. Der einzige Unterschied besteht darin, dass sich der verwendete Browser nach jedem Durchlauf des LoginBots schliesst und die Challenge-Seite bei einem erneuten Durchlauf neu geöffnet wird. Dies wurde deshalb so implementiert, da falls die MITM-Seite eingeschleust wurde, sich der Login-Bot nicht mehr anmelden kann. Dies würde dazu führen, dass keine weiteren Durchläufe mehr gestartet werden könnten.

6.5 Testkonzept

Beim Entwurf und der Planung des Testkonzepts wurde vor allem darauf geachtet, dass die Backend-Services der Architektur mit automatisierten Tests abgedeckt werden können. Für jede Teststufe wurde eine kurze Übersicht erstellt, welche die Testobjekte und die Testumgebung genauer beschreiben.

6.5.1 Komponententest

Testobjekte

Die Komponenten Tests werden für die RestAPI der Challenges sowie den CAS- und den C2-Server durchgeführt. Komponententests für den ChallengeManager werden nicht durchgeführt, da er keine Busienesslogik hat und nur für Weiterleitungen zuständig ist, dies kann mittels eines Integrationstests besser getestet werden.

Testumgebung

Die Tests können mittels Mocha [19], ein Testframework für JavaScript, automatisiert durchgeführt werden. Für Mocking und Stubbing wird das Sinon [20] verwendet.

Testübersicht

In der Tabelle 6.1 werden die durchgeführten Komponententests nochmals übersichtshalber aufgezeigt und beschrieben.

TABELLE 6.1: Übersicht der Komponententests.

Testobjekt	Beschreibung
RestAPI Easy Challenge	Test des Grundgerüsts, besonderes Augenmerk auf das Caching.
RestAPI Medium Challenge	Besonders das neu hinzugefügte Login musste getestet werden.
RestAPI Hard Challenge	Rückbau der Sicherheitslücken wurden getestet. Neue getLogin-Route musste getestet werden.
C2-Server	Startprozedur der Dockercontainer wurde getestet.
CAS-Server	Erstellung und handling der Token wurde getestet.
ChallengeManager	Keine Tests durchgeführt.

6.5.2 Integrationstest

Testobjekte

Beim Integrationstest werden die Frontend- sowie die Backend-Services getestet. Das bedeutet, es wurde die Integration der Challenge Webseiten mit ihren dazugehörigen RestAPI's getestet. Zusätzlich wurde die Integration der einzelnen Komponenten untereinander getestet.

Testumgebung

Um die Challenges lokal auszuführen, sind während der Entwicklung Konfigurationsanpassungen erforderlich. Dieser Prozess ist jedoch sehr fehleranfällig. Daher ist die lokale Umgebung nicht geeignet für Integrationstests. Stattdessen werden die Integrationstests auf dem ZHAW-Server durchgeführt. Dabei werden die Interaktionen der einzelnen Komponenten manuell getestet. Es wurden keine spezifischen Testanforderungen für die Tests erstellt.

Testübersicht

In der Tabelle 6.2 werden die durchgeführten Integrationstests übersichtshalber aufgelistet. Die Tabelle zeigt, welche Komponenten miteinander getestet wurden. Das Ziel der einzelnen Tests war es, die korrekten Interaktionen der Komponenten untereinander sicherzustellen.

TABELLE 6.2: Übersicht der Komponententests.

Testkomponente 1	Testkomponente 2
RestAPI Easy Challenge	Webseite Easy Challenge
RestAPI Medium Challenge	Webseite Medium Challenge
RestAPI Hard Challenge	Webseite Hard Challenge
Easy Challenge	CAS-Server
Medium Challenge	CAS-Server
Hard Challenge	CAS-Server
Easy Challenge	C2-Server
Medium Challenge	C2-Server
Hard Challenge	C2-Server
C2-Server	ChallengeManager
C2-Server	CAS-Server

6.5.3 Systemtest

Testobjekt

In dieser Testphase werden alle Systemkomponenten gemeinsam getestet. Das zu testende Objekt ist das Gesamtsystem, das aus den Challenges, dem C2-Server, dem CAS-Server und dem ChallengeManager besteht.

Testumgebung

Die laufende Version des Systems wird auf dem Server ZHAW getestet und wird manuell durch das Spielen der Challenges getestet.

7 Verifizierung und Evaluation

In Kapitel 6 wurde die Architektur und damit die realisierten Challenges beschrieben sowie die drei Server, die das zentrale System bilden. In diesem Kapitel werden diese Komponenten auf ihre Anforderungen (siehe Kapitel 4) überprüft. Es wird geschaut, ob alle gestellten Anforderungen umgesetzt wurden. Falls es Anforderungen gibt, die nicht umgesetzt werden konnten, wird analysiert, weshalb dies nicht geschehen konnte. Für die Verifizierung werden folgende Symbole verwendet:

✔: Symbolisiert, dass die Anforderung vollständig umgesetzt wurde.

✔: Symbolisiert, dass die Anforderung teilweise umgesetzt wurde oder nur teilweise umgesetzt werden konnte.

✖: Symbolisiert, dass die Anforderung nicht umgesetzt wurde oder nicht umgesetzt werden konnte.

7.1 Verifizierung der Anforderungen

Die nachfolgenden Tabellen fassen die Resultate der Anforderungen zusammen. Jede Tabelle ist dabei in drei Spalten aufgeteilt:

1. Spalte: ID, die auf die Anforderung referenziert.
2. Spalte: Fasst die Anforderung kurz zusammen.
3. Spalte: Hält fest, ob die Anforderung vollständig, teilweise oder nicht erfüllt wurde.

Tabelle 7.3 fasst die Resultate der nichtfunktionalen Anforderungen zusammen. Bei den nichtfunktionalen Anforderungen konnten alle Anforderungen erfolgreich umgesetzt werden.

TABELLE 7.1: Überprüfung der Erfüllung aller nichtfunktionalen Anforderungen.

Anforderung	Beschreibung	ist erfüllt
Req-NA-01	Erfahrungsniveau der Spieler*innen.	✔
Req-NA-02	Wissensvoraussetzung an die Challenges Easy und Medium.	✔
Req-NA-03	Ziel der Challenges.	✔
Req-NA-04	Spassfaktor der Challenges.	✔
Req-NA-05	Hints für Challenges Easy und Medium.	✔
Req-NA-06	Auslastung der Challenges.	✔
Req-NA-07	Realitätsbezug der Challenges.	✔
Req-NA-08	Realitätsbezug der Challenge Hard.	✔
Req-NA-09	Hints für die Challenge Hard.	✔

Tabelle 7.2 fasst die Resultate der technischen Anforderungen zusammen. Auch hier konnten alle Anforderungen erfolgreich erfüllt werden.

TABELLE 7.2: Überprüfung der Erfüllung aller technischen Anforderungen.

Anforderung	Beschreibung	ist erfüllt
Req-TA-01	Anzahl Schwierigkeitsstufen.	✔
Req-TA-01.Easy.1		✔
Req-TA-01.Medium.1		✔
Req-TA-01.Medium.2		✔
Req-TA-01.Hard.1		✔
Req-TA-01.Hard.2		✔
Req-TA-02	Brute-Force-Angriffea wird standgehalten.	✔
Req-TA-03	Mehrere Spieler*innen können die gleiche Challenge gleichzeitig spielen.	✔
Req-TA-04	Flag ist nur durch beabsichtigte Sicherheitslücke erhaltbar.	✔
Req-TA-05	Grundstruktur aller Challenges ist gleich.	✔
Req-TA-06	Erweiterbarkeit der Challenges.	✔

Req-TA-04 sagt, dass nur die beabsichtigte Sicherheitslücke einer Challenge ausgenutzt werden können soll, um an das Flag zu gelangen. Überprüfen, ob eine Software keine Sicherheitslücken besitzt, kann nicht immer zu 100 % bestätigt werden. Da jedoch das Flag nur erhaltbar ist, wenn es dem Bot entwendet wird, indem der Cache angegriffen wird, lassen sich andere Sicherheitslücken ausschlissen.

Tabelle 7.3 listet die Resultate der Anforderungen an die Interoperabilität auf. Diese konnten ebenfalls erfolgreich umgesetzt werden.

TABELLE 7.3: Überprüfung der Erfüllung der Anforderungen zur Einbindung in die ZHAW Umgebung.

Anforderung	Beschreibung	ist erfüllt
Req-IOP-01	Zentrales System steuert Erstellung der Challenges.	✔
Req-IOP-02	CTF Plattform verwaltet Punktevergabe.	✔
Req-IOP-03	Hints sind auf der CTF Plattfrom ersichtlich.	✔

Für Req-IOP-02 und Req-IOP-03 musste die Konfiguration der CTF Plattform angepasst werden. Das Anpassen dieser Konfiguration wurde von Thomas Sutter basierend auf einer Konfigurationsdatei durchgeführt. Die Konfigurationsdatei, ist im Anhang A.2 zu finden.

Tabelle 7.4 fasst die Resultate der Anforderung an die Sicherheit der Challenges zusammen. Diese Anforderungen konnten nur geringfügig erfüllt werden.

TABELLE 7.4: Überprüfung der Erfüllung der Anforderungen an die Sicherheit der Challenges.

Anforderung	Beschreibung	ist erfüllt
Req-S-01	Challenge vor Fremdeinwirkung schützen.	✔
Req-S-02	Erzeugung neuer Challenge-Umgebungen limitieren.	✖

Für Req-S-01 wurde der CAS-Server entwickelt. Dieser verhindert, dass fremde Spieler*innen Zugriff auf den Cache der eigenen Challenge haben. Dies trifft für die Challenge Easy und Medium zu. Die Challenge Hard implementiert diesen Prozess ebenfalls, jedoch ist der Varnish Cache nicht davon betroffen. D. h. ein*e Spieler*in mit bösartigen Absichten, kann andauernd Requests auf die Login-Seite absetzen. Dies führt dazu, dass der Varnish Cache nie eine bösartige Response im Cache speichert. Die Konsequenz ist, dass die Challenge nicht lösbar wird.

Req-S-02 konnte nicht umgesetzt werden. Dies, da ein*e Spieler*in nicht über einen Token im Vorhinein identifiziert werden kann. Dadurch kann ein*e Spieler*in den Challenge-Manager mit Anfragen überfluten und somit alle Ports besetzen, die dem C2-Server zur Verfügung stehen. Die Folge ist, dass ein DOS-Angriff auf den Challenge-Manager möglich ist.

7.2 Evaluation der Challenges Easy und Medium

An einem CTF-Event (siehe Abschnitt 1.5), der intern an der ZHAW durchgeführt wurde, konnten die ersten beiden Challenges überprüft werden.

Das Feedback für die Easy Challenge fiel grösstenteils positiv aus. Die einzigen Punkte, die bemängelt wurden, waren die Platzierung des Flags sowie die Burp-Anleitung im Tutorial. Diese Punkte wurden anschliessend nochmals überarbeitet.

Die Medium Challenge wurde generell als zu schwierig eingestuft. Daher wurden Hints zu der Challenge hinzugefügt, um den Spielenden dabei zu helfen, den richtigen Lösungsweg zu finden.

Die detaillierten Resultate der Umfrage können im Anhang A.3 betrachtet werden.

8 Ausblick

Nach erfolgreicher Umsetzung der Challenges und dem Erhalt von einem ersten Feedback während eines CTF-Events, bietet dieses Kapitel einen Ausblick auf zukünftige Verbesserungs- und Erweiterungsmöglichkeiten.

Grösseres Event: Die Anzahl an Rückmeldungen ist mit drei relativ gering und die letzte Challenge hat noch gar keine Rückmeldungen erhalten. Daher würde sich ein grösseres CTF Event bestens anbieten, um mehr Feedback einzuholen. Ebenfalls kann dabei die Performance des zentralen Systems besser getestet werden, als es im kleinen Rahmen möglich war.

Neue Challenges: Die drei entwickelten Challenges erstrecken sich über ein breites Schwierigkeitsspektrum. Für alle Erfahrungsstufen (Einsteiger*in, Intermediate und Expterte*Expertin) gibt es eine Challenge jedoch noch zu wenige, um seine Fähigkeiten gezielt zu schärfen. Für alle drei Schwierigkeitsstufen (Easy, Medium und Hard) können weitere Challenges entwickelt werden, um den Erwartungen des jeweiligen Publikums gerecht zu werden. Die entwickelte Struktur ermöglich ein einfaches integrieren in die CTF Plattform. Bei Bedarf können die neuen Challenges auch komplett von der Struktur der jetzigen Challenges abweichen.

Sicherung der Challenges: Aktuell kann von CTFd kein Token für angemeldete Benutzer*innen erstellt werden. Falls sich in Zukunft über CTFd ein Token generieren lässt, kann der CAS-Server und die damit verbundenen Prozesse angepasst werden. Die bisherige Implementation des Token-Managements ist sehr einfach gehalten und schränkt auch die Spieler*innen in gewisser Weise ein. Die dadurch erhaltene Sicherheit ist notwendig, jedoch nicht optimal.

Sicherung vor DOS-Angriffen: Wie in Kapitel 7 beschreiben, sind DOS-Angriffe auf den Challenge-Manager möglich. Durch das Einführen eines Tokens, welcher von der CTF Plattform der ZHAW stammt, wäre es möglich, mehrere Anfragen der gleichen Person zu erkennen und zu unterbinden.

9 Konklusion

Das Ziel dieser Arbeit bestand darin, eigene Web Cache Poisoning CTF Challenges für die CTF Plattform der ZHAW zu entwickeln. Zum erreichen dieses Zieles, wurde zu Beginn eine Marktanalyse durchgeführt, um herauszufinden, wie andere Hochschulen CTF Challenges im Bereich der Higher Education nutzen und wie diese aufgebaut sind. Basierend auf den gewonnenen Erkenntnissen wurden die Anforderungen und das Konzept für die Challenges erarbeitet. Anschliessend wurde die Architektur der Challenges unter Berücksichtigung dieser Anforderungen und des Konzepts geplant und umgesetzt.

Es konnten erfolgreich drei Challenges umgesetzt werden. Die erste Challenge dient als Tutorial, bei dem die Grundlagen des Web Cache Poisoning vermittelt werden. Dabei können erste Erfahrungen gesammelt werden. Bei der zweiten Challenge wurde die Schwierigkeitsstufe erhöht, indem sowohl Web Cache Poisoning als auch XSS eingesetzt werden müssen, um ein Cookie eines Bots zu stehlen. Die letzte und anspruchsvollste Challenge basiert auf einer echten Sicherheitslücke, die in Drupal entdeckt wurde. Dabei muss die Login-Seite der Challenge mit einer MITM-Seite ersetzt werden, um die Logindaten des Bots zu stehlen. Somit stehen nun drei Web Cache Poisoning CTF Challenges auf der CTF Plattform der ZHAW zur Verfügung.

Durch die Durchführung eines kleinen CTF-Events konnten wertvolle Rückmeldungen zu den ersten beiden Challenges gesammelt werden. Anhand dieses Feedbacks war es möglich, die Challenges weiter zu optimieren und zu verbessern.

Für die Verteilung und Sicherung der Challenges wurden verschiedene Mechanismen implementiert. Zum Spielen der Challenges wurde ein eigenes System entwickelt, welches die virtuellen Umgebungen der Challenges verwaltet. Um die Challenges vor Fremdeinwirkungen zu schützen, wurde ein CAS-Server entwickelt. Dieser verhindert, dass Unbefugte Zugriff auf die Challenges erhalten. Eine vollständige Sicherung der Challenges konnte nicht komplett umgesetzt werden. Solange CTFd kein Token für angemeldete Benutzer*innen zur Verfügung stellen kann, wird dies wohl auch nicht möglich sein.

Bei der Implementierung und Planung der Challenges wurde besonderer Wert darauf gelegt, dass sowohl die Challenges selbst als auch die Verteilungs- und Sicherheitsmechanismen leicht erweiterbar sind. Dies ermöglicht eine einfache Integration neuer Challenges in diese Umgebung.

In Zukunft besteht noch ausreichend Raum für die Erstellung weiterer Web Cache Poisoning CTF Challenges. Die in dieser Arbeit entwickelten Challenges können dabei als Vorlage dienen. Darüber hinaus können auch die Verteilungs- und Sicherheitsmechanismen für zukünftige Challenges genutzt werden, unabhängig davon, ob es sich um Web Cache Poisoning Challenges handelt oder nicht.

Literaturverzeichnis

[1] CTFd LLC. *Cyber Security Training made simple*. [Stand: 31.5.2023]. URL: https://ctfd.io/.

[2] Arvind S. Raj u. a. „Scalable and Lightweight CTF Infrastructures Using Application Containers (Pre-recorded Presentation)“. In: *2016 USENIX Workshop on Advances in Security Education (ASE 16)*. Austin, TX: USENIX Association, Aug. 2016. URL: https://www.usenix.org/conference/ase16/workshop-program/presentation/raj.

[3] Menelaos Katsantonis, Panayotis Fouliras und Ioannis Mavridis. „Conceptual analysis of cyber security education based on live competitions“. In: *2017 IEEE Global Engineering Education Conference (EDUCON)*. 2017, S. 771–779. DOI: 10.1109/EDUCON.2017.7942934.

[4] Adrian Dabrowski u. a. „Leveraging Competitive Gamification for Sustainable Fun and Profit in Security Education“. In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. Washington, D.C.: USENIX Association, Aug. 2015. URL: https://www.usenix.org/conference/3gse15/summit-program/presentation/dabrowski.

[5] Shellphish. *iCTF: the International Capture The Flag Competition*. [Stand: 1.6.2023]. URL: https://shellphish.net/ictf/.

[6] James Kettle. *Practical Web Cache Poisoning: Redefining 'Unexploitable'*. Techn. Ber. Aug. 2018.

[7] Cloudflare Docs. *Cache Keys*. [Stand: 14. April 2023]. URL: https://developers.cloudflare.com/cache/about/cache-keys.

[8] The MITRE Corporation. *CVE-2018-14773*. [Stand: 31.5.2023]. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-14773.

[9] Zend Technologies. *ZF2018-01: URL Rewrite vulnerability*. [Stand: 31.5.2023]. URL: https://framework.zend.com/security/advisory/ZF2018-01.

[10] Drupal Security Team. *CVE-2018-14773*. [Stand: 31.5.2023]. URL: https://www.drupal.org/SA-CORE-2018-005.

[11] James Kettle. *Web Cache Entanglement: Novel Pathways to Poisoning*. Techn. Ber. Aug. 2020.

[12] Cloudflare Docs. *Default Cache Behaviour: Cloudflare cache responses*. [Stand: 14. April 2023]. URL: https://developers.cloudflare.com/cache/about/default-cache-behavior/#cloudflare-cache-responses.

[13] Cloudflare Docs. *Avoid Web Cache Poisoning*. [Stand: 14. April 2023]. URL: https://developers.cloudflare.com/cache/best-practices/avoid-web-poisoning.

[14] HTTP Working Group. *HTTP Semantics*. [Stand: 14. April 2023]. URL: https://httpwg.org/http-core/draft-ietf-httpbis-semantics-latest.html#GET.

[15] Port Swigger. *Param Miner*. [Stand: 6.6.2023]. URL: https://portswigger.net/bappstore/17d2949a985c4b7ca092728dba871943.

[16] Impulse Communications. *Fake People*. [Stand: 1.6.2023]. URL: https://boredhumans.com/faces.php/.

[17] OpenJS Foundation. *Fast, unopinionated, minimalist web framework for Node.js*. [Stand: 6.6.2023]. URL: https://expressjs.com/.

[18] Pug. *A high-performance template engine*. [Stand: 6.6.2023]. URL: https://pugjs.org/api/getting-started.html.

[19] OpenJS Foundation. *Mocha: simple, flexible fun*. [Stand: 7.6.2023]. URL: https://mochajs.org/.

[20] Sinon.JS. *Standalone test spies, stubs and mocks for JavaScript*. [Stand: 7.6.2023]. URL: https://sinonjs.org/.

[21] Software Freedom Conservancy. *Selenium automates browsers. That's it!* [Stand: 6.6.2023]. URL: https://www.selenium.dev/documentation/webdriver/.

[22] Mathias Peter. *Simple and fast NodeJS internal caching*. [Stand: 8.6.2023]. URL: https://github.com/node-cache/node-cache.

[23] Dries Buytaert. *Web Server (Proxy) Caching*. [Stand: 6.6.2023]. URL: https://www.drupal.org/docs/7/managing-site-performance-and-scalability/caching-to-improve-performance/caching-overview#s-web-server-proxy-caching.

[24] Varnish Software. *Cookies*. [Stand: 6.6.2023]. URL: https://varnish-cache.org/docs/trunk/users-guide/increasing-your-hitrate.html#cookies.

[25] Varnish Software. *Authorization*. [Stand: 6.6.2023]. URL: https://varnish-cache.org/docs/trunk/users-guide/increasing-your-hitrate.html#authorization.

Abbildungsverzeichnis

1.1 Zeitplan für die Bachelorarbeit. 2

2.1 Aufbau einer CTF-Challenge mittels CTFd. 6

3.1 Methodik eines Web Cache Poisoning Angriffs [11]. 9

5.1 Mockup der Homepage. 22
5.2 Mockup der Detail-Ansicht. 22
5.3 Mockup der Tutorial-Seite. 23

6.1 Einbindung des Challenge Manager und C2-Servers in die ZHAW CTFd Infrastruktur. 25
6.2 Ablauf beim Zugreifen auf eine Challenge. 26
6.3 Mockup der Landing Page des Challenge Managers für die Challenge Medium. 26
6.4 Einbindung des CAS-Servers. 27
6.5 Verifizierung des Tokens beim Spielen einer Challenge. 28
6.6 Vereinfachte Darstellung der Drei-Schichten-Architektur. 30
6.7 Darstellung der Controller-Schicht des allgemeinen Aufbaus. 30
6.8 Darstellung der Service-Schicht des allgemeinen Aufbaus. 31
6.9 Darstellung der Model-Schicht des allgemeinen Aufbaus. 31
6.10 Funktionsweise des entwickelten Frameworks für zur Kommunikation mit der RestAPI. 32
6.11 Allgemeine Struktur einer Seite. 33
6.12 Struktur des Containers der Challenge How To Web Cache Poison. . . 34
6.13 Struktur des Containers der Challenge Cross Site Cookie Mining. . . . 36
6.14 Darstellung der Controller-Schicht der Medium Challenge. 37
6.15 Kommunikation zwischen Server und Webseite des Catchers. 39
6.16 Struktur des Containers der Challenge Double Cache Mayhem. 42
6.17 Darstellung der Controller-Schicht der Medium Challenge. 43
6.18 Verwendete JavaScript Files der MITM Home-Seite. 45

Tabellenverzeichnis

4.1 Nichtfunktionale Anforderungen der CTF Challenges. 13
4.2 Technische Anforderungen an die CTF-Challenges. 14
4.3 Spezifizierung der Anforderung Req-TA-01 für die Challenge Easy. . . 14
4.4 Spezifizierung der Anforderung Req-TA-01 für die Challenge Medium. 14
4.5 Spezifizierung der Anforderung Req-TA-01 für die Challenge Hard. . . 14
4.6 Anforderungen zur Interoperabilität der Challenges mit der ZHAW Umgebung. 15
4.7 Anforderungen an die Sicherung der Challenges. 15

6.1 Übersicht der Komponententests. 47
6.2 Übersicht der Komponententests. 48

7.1 Überprüfung der Erfüllung aller nichtfunktionalen Anforderungen. . . 49
7.2 Überprüfung der Erfüllung aller technischen Anforderungen. 50
7.3 Überprüfung der Erfüllung der Anforderungen zur Einbindung in die ZHAW Umgebung. 50
7.4 Überprüfung der Erfüllung der Anforderungen an die Sicherheit der Challenges. 51

Codeverzeichnis

3.1 Einfaches Beispiel eines Cache-Keys. 8
3.2 Einfaches Beispiel von Web Cache Poisoning. 8
3.3 Cachebuster im Einsatz. 11
3.4 Erweitertes Beispiel, wie der Cachebuster verwendet werden kann. . . 11
5.1 Infizieren des Caches der Easy Challenge. 18
5.2 Infizieren des Caches der Medium Challenge. 19
5.3 Infizieren des internen Caches. 20
5.4 Infizieren des externen Caches. 21
6.1 Funktionsbeschreibung der GET-Funktion. 33
6.2 Sicherheitslücke im Programmfluss des LessonControllers. 35
6.3 Sicherheitslücke in der createSingleLesson-Funktion des singleLesson.js Files. 35
6.4 Sicherheitslücke in der getById-Funktion des LessonControllers. 38
6.5 Sicherheitslücke in der Konfiguration des Session-Cookies. 38
6.6 Implementierung des Socket-Listeners auf der Webseite. 40
6.7 Konfiguration des Login-Bots zur Ausführung in einem Container. . . 41
6.8 Sicherheitslücke in der generateCacheKey-Funktion des CacheHandler. 43

A Anhang

A.1 API-Documentation

API Reference

Web Cache Poisoning API

API Version: 1.0.0

This document contains information about all available Routes, across all the currently available challenges.

INDEX

1. GET 3

1.1 GET /api/lessons 3

1.2 GET /api/lessons/{id} 3

1.3 GET /api/getlogin{destination} 4

2. POST 6

2.1 POST /api/login/ 6

2.2 POST /api/logout/ 6

API

1. GET

All GET requests provided by the API.

1.1 GET /api/lessons

Get all lessons currently available.

Get all lessons currently available from the database.

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Successful operation. All available lessons were retrieved.

RESPONSE MODEL - application/json

```
[{
Array of object:
  id        number
  title     string
  image     string
  subjects  [string]
}]
```

STATUS CODE - 404: Could not find any lessons.

RESPONSE MODEL - application/json

```
{
  Message  string
}
```

STATUS CODE - 500: Error occured while getting the lessons.

RESPONSE MODEL - application/json

```
{
  Message  string
}
```

1.2 GET /api/lessons/{id}

Get a single lesson.

Get a single lesson from the Database.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*id	number		ID of the lesson to get.

RESPONSE

STATUS CODE - 200: Successful operation. The requested lesson will be returned.

RESPONSE MODEL - application/json

```
{
  id        number
  title     string
  image     string
  subjects  [string]
  contact   string
}
```

STATUS CODE - 404: Could not find any lessons.

RESPONSE MODEL - application/json

```
{
  Message  string
}
```

STATUS CODE - 500: Error occured while getting the lesson.

RESPONSE MODEL - application/json

```
{
  Message  string
}
```

1.3 GET /api/getlogin{destination}

Get the location of the login page.

Gets the location of the login page and returns it.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
destination	string		Optional destination parameter. Can be used to set the destination of the login page.

RESPONSE

STATUS CODE - 200: Successful operation.

RESPONSE MODEL - application/json

```
{
  location  string
}
```

STATUS CODE - 500: Error occured while getting the lessons.

RESPONSE MODEL - application/json

```
{
  Message  string
}
```

2. POST

All POST requests provided by the API.

2.1 POST /api/login/

Handles a Login request.

Handles the login of the user and sets the login cookie. It will return a 401 Unauthorized, when the credentials are invalid.

REQUEST

REQUEST BODY - application/json

```
{
  user     string
  password string
}
```

RESPONSE

STATUS CODE - 204: Login was successfull. The session ID is returned in a cookie named `connect.sid`. You need to include this cookie in subsequent requests.

STATUS CODE - 401: Unauthorized, invalid username or password were provided.

RESPONSE MODEL - application/json

```
{
  Message string
}
```

STATUS CODE - 500: Error occured while logging in the user.

RESPONSE MODEL - application/json

```
{
  Message string
}
```

2.2 POST /api/logout/

Handles a logout request.

Logout of the user will be executed and the logout cookie wil be set.

REQUEST

No request parameters

RESPONSE

STATUS CODE - 204: Logout was successfull.

STATUS CODE - 500: Error occured while logging out the user.

RESPONSE MODEL - application/json

```
{
  Message  string
}
```

A.2 Konfiguration für die Challenges auf der CTFd Plattform der ZHAW

Challenge: Easy

Titel	How to Web Cache Poison
Adresse	[redacted]
Beschreib	Do you want to Web Cache Poison? If so, try this beginner guide and get your first hands on experience.
Flag	[redacted]
Hints	keine
Punkteabzug	keinen
Punkte	100

Challenge: Medium

Titel	Cross Site Cookie Mining
Adresse	[redacted]
Beschreib	Mining cookies with Web Cache Poisoning? Seems like fun.
Flag	[redacted]
Hints	• Param Mining was mentioned in the Tutorial, maybe have a look at it again. • Maybe XSS can be combined with Cache Poisoning. • Cookies sometimes contain more information than they should. • Catching sites can be useful to get information of other users.
Punkteabzug	keinen
Punkte	300

Challenge: Hard

Titel	Double Cache Mayhem
Adresse	[redacted]
Beschreib	Get ready for twice the mischief and double the excitement! If you're craving an extra dose of adventure, look no further because we've got you covered.
Flag	[redacted]
Hints	• The login could be a place worth investigating. • Multiple caches may exist. • Multiple steps are required. • CVE-2018-14773
Punkteabzug	-250 Punkte für den letzten Hinweis.
Punkte	500

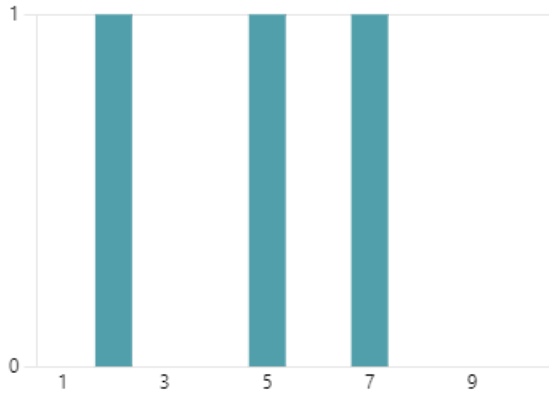
A.3 Resultate der Umfrage

Umfrage Web Cache Poisoning Challenges (EN)

3	03:15	Aktiv
Antworten	Durchschnittliche Zeit für das Ausfüllen	Status

1. How much experience do you have in CTF's?

4.67
Durchschnittliche Bewertung



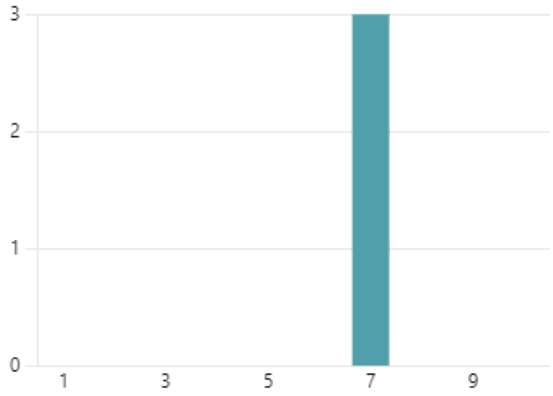
2. Did you try to solve the Easy Challenge (How To Web Cache Poison)?

Yes	3
No	0

Umfrage Web Cache Poisoning Challenges (EN)

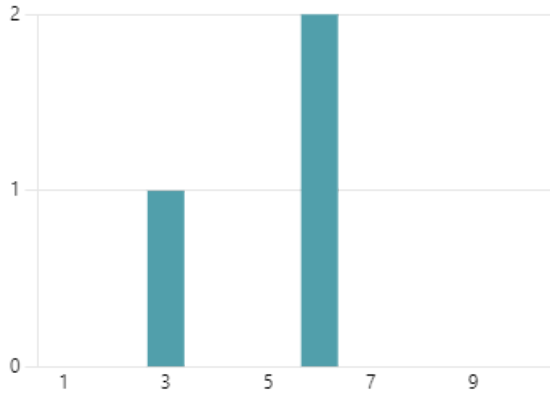
3. Easy Challenge: How much fun did you have solving the challenge?

7.00
Durchschnittliche Bewertung



4. Easy Challenge: How difficult was the challenge?

5.00
Durchschnittliche Bewertung



5. Was the tutorial clear and understandable? Did the theory part help?

3
Antworten

Neueste Antworten

"*Theory part was very good. As for the tutorial make sure that everyone understands how to set up burp with a proxy.*"

"*could be clearer for burp-newbies*"

"*yes i did help*"

6. Would you recommend the challenge?

7. If you do not recommend the challenge, could you please elaborate why?

0
Antworten

Neueste Antworten

8. Feedback for the Easy Challenge:

3
Antworten

Neueste Antworten

"*Make it clear that the flag is in the tutorial description and not on the webpage. *"

"*having the flag in the alert would be nice*"

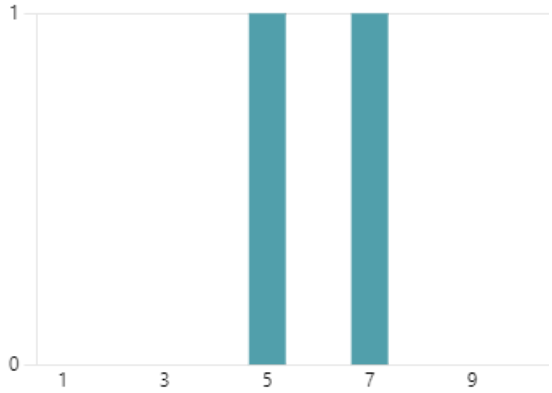
"."

Umfrage Web Cache Poisoning Challenges (EN)

9. Did you try to solve the Medium Challenge (Cross Site Cookie Mining)?

10. Medium Challenge: How much fun did you have solving the challenge?

6.00
Durchschnittliche Bewertung



11. Medium Challenge: Empfundener Schwierigkeitsgrad

10.00
Durchschnittliche Bewertung

12. Was the information provided enough to solve the Medium Challenge?

2
Antworten

Neueste Antworten
"*not for me*"
"*i couldn't solve it*"

13. Would you recommend the Medium Challenge?

14. If you do not recommend the challenge, could you please elaborate why?

2
Antworten

Neueste Antworten
"*I cannot really answer this question, since I could not solve it*"
"*it's hard*"

15. Feedback for the Medium Challenge:

2
Antworten

Neueste Antworten
"*hints fot newbies wären nützlich.*"
" ."